A Lambda-Calculus Foundation for Universal Probabilistic Programming

A Lambda-Calculus Foundation for Universal Probabilistic Programming *

Johannes Borgström
Uppsala University
Sweden

Ugo Dal Lago
University of Bologna, Italy &
INRIA, France

Andrew D. Gordon
Microsoft Research &
University of Edinburgh
United Kingdom

Marcin Szymczak
University of Edinburgh
United Kingdom

Abstract

We develop the operational semantics of an untyped probabilistic λ-calculus with continuous distributions, and both hard and soft constraints, as a foundation for universal probabilistic programming languages such as CHURCH, ANGLICAN, and VENTURE. Our first contribution is to adapt the classic operational semantics of λ-calculus to a continuous setting via creating a measure space on terms and defining step-indexed approximations. We prove equivalence of big-step and small-step formulations of this *distribution-based semantics*. To move closer to inference techniques, we also define the *sampling-based semantics* of a term as a function from a trace of random samples to a value. We show that the distribution induced by integration over the space of traces equals the distribution-based semantics. Our second contribution is to formalize the implementation technique of trace *Markov chain Monte Carlo* (MCMC) for our calculus and to show its correctness. A key step is defining sufficient conditions for the distribution induced by trace MCMC to converge to the distribution-based semantics. To the best of our knowledge, this is the first rigorous correctness proof for trace MCMC for a higher-order functional language, or for a language with soft constraints.

Categories and Subject Descriptors D.3.1 [*Programming Languages*]: Formal Definitions and Theory—Semantics; F.3.2 [*Logic and Meaning of Programs*]: Semantics of Programming Languages—Operational Semantics; G.3 [*Probability and Statistics*]: Probabilistic algorithms (including Monte Carlo)

General Terms Algorithms, Languages

Keywords Probabilistic Programming, Lambda-calculus, MCMC, Machine Learning, Operational Semantics

* The first author is supported by the Swedish Research Council grant 2013-4853. The second author is partially supported by the ANR project 12IS02001 PACE and the ANR project 14CE250005 ELICA. The fourth author was supported by Microsoft Research through its PhD Scholarship Programme.

1. Introduction

In computer science, probability theory can be used for models that enable system abstraction, and also as a way to compute in a setting where having access to a source of randomness is essential to achieve correctness, as in randomised computation or cryptography (Goldwasser and Micali 1984). Domains in which probabilistic models play a key role include robotics (Thrun 2002), linguistics (Manning and Schütze 1999), and especially machine learning (Pearl 1988). The wealth of applications has stimulated the development of concrete and abstract programming languages, that most often are extensions of their deterministic ancestors. Among the many ways probabilistic choice can be captured in programming, a simple one consists in endowing the language of programs with an operator modelling the sampling from (one or many) distributions. This renders program evaluation a probabilistic process, and under mild assumptions the language becomes universal for probabilistic computation. Particularly fruitful in this sense has been the line of work in the functional paradigm.

In *probabilistic programming*, programs become a way to specify probabilistic models for observed data, on top of which one can later do inference. This has been a source of inspiration for AI researchers, and has recently been gathering interest in the programming language community (see Goodman (2013), Gordon et al. (2014), and Russell (2015)).

1.1 Universal Probabilistic Programming in CHURCH

CHURCH (Goodman et al. 2008) introduced *universal probabilistic programming*, the idea of writing probabilistic models for machine learning in a Turing-complete functional programming language. CHURCH, and its descendants VENTURE (Mansinghka et al. 2014), ANGLICAN (Tolpin et al. 2015), and WEB CHURCH (Goodman and Tenenbaum 2014) are dialects of SCHEME. Another example of universal probabilistic programming is WEBPPL (Goodman and Stuhlmüller 2014), a probabilistic interpretation of JAVASCRIPT.

A probabilistic query in CHURCH has the form:

```
(query (define x1 e1)...(define xn en) eq ec)
```

The query denotes the distribution given by the probabilistic expression e_q, given variables x_i defined by potentially probabilistic expressions e_i, constrained so that the boolean predicate e_c is true.

Consider a coin with bias p, that is, p is the probability of heads. Recall that the *geometric distribution* of the coin is the distribution over the number of flips in a row before it comes up heads. An example of a CHURCH query is as follows: it denotes the geometric distribution for a fair coin, constrained to be greater than one.

```
(query
  (define flip (lambda (p) (< (rnd) p)))
  (define geometric (lambda (p)
    (if (flip p) 0 (+ 1 (geometric p)))))
  (define n (geometric .5))
  n
  (> n 1))
```

The query defines three variables: (1) `flip` is a function that flips a coin with bias `p`, by calling `(rnd)` to sample a probability from the uniform distribution on the unit interval; (2) `geometric`[1] is a function that samples from the geometric distribution of a coin with bias `p`; and (3) `n` denotes the geometric distribution with bias `0.5`. Here are samples from this query:

```
(5 5 5 4 2 2 2 2 2 3 3 2 2 7 2 2 3 4 2 3)
```

This example is a discrete distribution with unbounded support (any integer greater than one may be sampled with some non-zero probability), defined in terms of a continuous distribution (the uniform distribution on the unit interval). Queries may also define continuous distributions, such as regression parameters.

1.2 Problem 1: Semantics of CHURCH Queries

The first problem we address in this work is to provide a formal semantics for universal probabilistic programming languages with constraints. Our example illustrates the common situation in machine learning that models are based on continuous distributions (such as `(rnd)`) and use constraints, but previous works on formal semantics for untyped probabilistic λ-calculi do not rigorously treat the combination of these features.

To address the problem we introduce a call-by-value λ-calculus with primitives for random draws from various continuous distributions, and primitives for both hard and soft constraints. We present an encoding of CHURCH into our calculus, and some nontrivial examples of probabilistic models.

We consider two styles of operational semantics for our λ-calculus, in which a term is interpreted in two ways, the first closer to inference techniques, the second more extensional:

Sampling-Based: A function from a trace to a value and weight.

Distribution-Based: A distribution over terms of our calculus.

To obtain a thorough understanding of the semantics of the calculus, for each of these styles we present two inductive definitions of operational semantics, in small-step and big-step style.

First, we consider the *sampling-based semantics*: the two inductive definitions have the forms shown below, where M is a closed term, s is a finite *trace* of random real numbers, $w > 0$ is a *weight* (to impose soft constraints), and G is a *generalized value* (either a value (constant or λ-abstraction) or the exception `fail`, used to model a failing hard constraint).

- Figure 4 defines small-step relation $(M, w, s) \to (M', w', s')$.
- Figure 1 defines the big-step relation $M \Downarrow_w^s G$.

For example, if M is the λ-term for our geometric distribution example and we have $M \Downarrow_w^s G$ then there is $n \geq 0$ such that:

- the trace has the form $s = [q_1, \ldots, q_{n+1}]$ where each q_i is a probability, and $q_i < 0.5$ if and only if $i = n + 1$. (A sample $q_i \geq 0.5$ is tails; a sample $q_i < 0.5$ is heads.)
- the result takes the form $G = n$ if $n > 1$, and otherwise $G =$ `fail` (the failure of a hard constraint leads to `fail`);
- and the weight is $w = 1$ (the density of the uniform distribution on the unit interval).

Our first result, Theorem 1, shows equivalence: that the big-step and small-semantics of a term consume the same traces to produce the same results with the same weights.

To interpret these semantics probabilistically, we describe a metric space of λ-terms and let $\mathscr{D}$ range over *distributions*, that is, sub-probability Borel measures on terms of the λ-calculus. We define $[\![M]\!]_{\mathbb{S}}$ to be the distribution induced by the sampling-based semantics of M, by integrating the weight over the space of traces.

Second, we consider the *distribution-based semantics*, that directly associate distributions with terms, without needing to integrate out traces. The two inductive definitions have the forms shown below, where n is a step-index:

- Figure 6 defines a family of small-step relations $M \Rightarrow_n \mathscr{D}$.
- Figure 7 defines a family of big-step relations $M \Downarrow_n \mathscr{D}$.

These step-indexed families are approximations to their suprema, distributions written as $[\![M]\!]_{\Rightarrow}$ and $[\![M]\!]_{\Downarrow}$. By Theorem 2 we have $[\![M]\!]_{\Rightarrow} = [\![M]\!]_{\Downarrow}$. The proof of the theorem needs certain properties (Lemmas 12, 15, and 17) that build on compositionality results for sub-probability kernels (Panangaden 1999) from the measure theory literature. We apply the distribution-based semantics in Section 4.7 to show an equation between hard and soft constraints.

Finally, we reconcile the two rather different styles of semantics: Theorem 3 establishes that $[\![M]\!]_{\mathbb{S}} = [\![M]\!]_{\Rightarrow}$.

1.3 Problem 2: Correctness of Trace MCMC

The second problem we address is implementation correctness. As recent work shows (Hur et al. 2015; Kiselyov 2016), subtle errors in inference algorithms for probabilistic languages are a motivation for correctness proofs for probabilistic inference.

Markov chain Monte Carlo (MCMC) is an important class of inference methods, exemplified by the Metropolis-Hastings (MH) algorithm (Metropolis et al. 1953; Hastings 1970), that accumulates samples from a target distribution by exploring a Markov chain generated from a proposal kernel Q. The original work on CHURCH introduced the implementation technique called *trace MCMC* (Goodman et al. 2008). Given a closed term M, trace MCMC generates a Markov chain of traces, s_0, s_1, s_2, ...

Our final result, Theorem 4, asserts that the Markov chain generated by trace MCMC for a particular choice of Q converges to a stationary distribution, and that the induced distribution on values is equal to the semantics $[\![M]\!]_{\Rightarrow}$ conditional on success, that is, that the computation terminates and yields a value (not `fail`). We formalize the algorithm rigorously, and show that the resulting Markov chain satisfies standard criteria: *aperiodicity* and *irreducibility*. Hence, Theorem 4 follows from a classic result of Tierney (1994) together with Theorem 3.

1.4 Contributions of the Paper

We make the following original contributions:

1. Definition of an untyped λ-calculus with continuous distributions capable of encoding the core of CHURCH.
2. Development of both sampling-based and distribution-based semantics, shown equivalent (Theorems 1, 2, and 3).
3. First proof of correctness of trace MCMC for a λ-calculus (Theorem 4).

The only previous work on formal semantics of λ-calculi with constraints and continuous distributions is recent work by Staton et al. (2016). Their main contribution is an elegant denotational

[1] See `http://forestdb.org/models/geometric.html`.

semantics for a simply typed λ-calculus with continuous distributions and both hard and soft constraints, but without recursion. They do not consider MCMC inference. Their work does not apply to the recursive functions (such as the geometric distribution in Section 1.1) or data structures (such as lists) typically found in CHURCH programs. For our purpose of conferring formal semantics on CHURCH-family languages, we consider it advantageous to rely on untyped techniques.

The only previous work on correctness of trace MCMC, and an important influence on our work, is a recent paper by Hur et al. (2015) which proves correct an algorithm for computing an MH Markov chain. Key differences are that we work with higher-order languages and soft constraints, and that we additionally give a proof that our Markov chain always converges, via the correctness criteria of Tierney (1994).

An extended version (Borgström et al. 2015) of this paper includes detailed proofs.

2. A Foundational Calculus for CHURCH

In this section, we describe the syntax of our calculus and equip it with an intuitive semantics relating program outcomes to the sequences of random choices made during evaluation. By translating CHURCH constructs to this calculus, we show that it serves as a foundation for Turing-complete probabilistic languages.

2.1 Syntax of the Calculus

We represent scalar data as real numbers $c \in \mathbb{R}$. We use 0 and 1 to represent `false` and `true`, respectively. Let $\mathcal{I}$ be a countable set of *distribution identifiers* (or simply *distributions*). Metavariables for distributions are D, E. Each distribution identifier D has an integer *arity* $|\mathsf{D}| \geq 0$, and defines a density function $\mathrm{pdf}_{\mathsf{D}} : \mathbb{R}^{|\mathsf{D}|+1} \to [0, \infty)$ of a sub-probability kernel. For example, a draw ($\mathsf{rnd}()$) from the uniform distribution on the unit interval has density $\mathrm{pdf}_{\mathsf{rnd}}(c) = 1$ if $c \in [0, 1]$ and otherwise 0, while a draw ($\mathsf{Gaussian}(m, v)$) from the Gaussian distribution with mean m and variance v has density $\mathrm{pdf}_{\mathsf{Gaussian}}(m, v, c) = 1/(e^{\frac{(c-m)^2}{2v}}\sqrt{2v\pi})$ if $v > 0$ and otherwise 0.

Let g be a metavariable ranging over a countable set of *function identifiers* each with an integer *arity* $|g| > 0$ and with an interpretation as a total measurable function $\sigma_g : \mathbb{R}^{|g|} \to \mathbb{R}$. Examples of function identifiers include addition $+$, comparison $>$, and equality $=$; they are often written in infix notation. We define the *values* V and *terms* M as follows, where x ranges over a denumerable set of variables $\mathcal{X}$.

$$V ::= c \mid x \mid \lambda x.M$$

$$M, N ::= V \mid M\ N \mid \mathsf{D}(V_1, \ldots, V_{|\mathsf{D}|}) \mid g(V_1, \ldots, V_{|g|}) \mid \texttt{if}\ V\ \texttt{then}\ M\ \texttt{else}\ N \mid \texttt{score}(V) \mid \texttt{fail}$$

The term `fail` acts as an exception and models a failed hard constraint. The term $\texttt{score}(c)$ models a soft constraint, and is parametrized on a positive probability $c \in (0, 1]$. As usual, free occurrences of x inside M are bound by $\lambda x.M$. Terms are taken modulo renaming of bound variables. Substitution of all free occurrences of x by a value V in M is defined as usual, and denoted $M\{V/x\}$. Let Λ denote the set of all terms, and $C\Lambda$ the set of *closed* terms. The set of all *closed values* is $\mathcal{V}$, and we write $\mathcal{V}_\lambda$ for $\mathcal{V} \setminus \mathbb{R}$. *Generalized values* G, H are elements of the set $\mathcal{GV} = \mathcal{V} \cup \{\texttt{fail}\}$, *i.e.*, generalized values are either values or `fail`. Finally, *erroneous redexes*, ranged over by metavariables like T, R, are closed terms in one of the following five forms:

- $c\ M$.
- $\mathsf{D}(V_1, \ldots, V_{|\mathsf{D}|})$ where at least one of the V_i is a λ-abstraction.
- $g(V_1, \ldots, V_{|g|})$ where at least one of the V_i is a λ-abstraction.
- $\texttt{if}\ V\ \texttt{then}\ M\ \texttt{else}\ N$, where V is neither `true` nor `false`.
- $\texttt{score}(V)$, where $V \notin (0, 1]$.

2.2 Big-step Sampling-based Semantics

In defining the first semantics of the calculus, we use the classical observation (Kozen 1979) that a probabilistic program can be interpreted as a deterministic program parametrized by the sequence of random draws made during the evaluation. We write $M \Downarrow_w^s V$ to mean that evaluating M with the outcomes of random draws as listed in the sequence s yields the value V, together with the *weight* w that expresses how likely this sequence of random draws would be if the program was just evaluated randomly. Because our language has continuous distributions, w is a probability *density* rather than a probability mass. Similarly, $M \Downarrow_w^s \texttt{fail}$ means that evaluation of M with the random sequence s fails. In either case, the finite trace s consists of exactly the random choices made during evaluation, with no unused choices permitted.

Formally, we define *program traces* s, t to be finite sequences $[c_1, \ldots, c_n]$ of reals of arbitrary length. We let $M \Downarrow_w^s G$ be the least relation closed under the rules in Figure 1. The (EVAL RANDOM) rule replaces a random draw from a distribution D parametrized by a vector $\vec{c}$ with the first (and only) element c of the trace, presumed to be the outcome of the random draw, and sets the weight to the value of the density of $\mathsf{D}(\vec{c})$ at c. (EVAL RANDOM FAIL) throws an exception if c is outside the support of the corresponding distribution. Meanwhile, (EVAL SCORE), applied to $\texttt{score}(c)$, sets the weight to c and returns a dummy value. The applications of soft constraints using `score` are described in Section 2.5.

All the other rules are standard for a call-by-value lambda-calculus, except that they allow the traces to be split between subcomputations and they multiply the weights yielded by subcomputations to obtain the overall weight.

2.3 Encoding CHURCH

We now demonstrate the usefulness and expressive power of the calculus via a translation of CHURCH, an untyped higher-order functional probabilistic language.

The syntax of CHURCH's *expressions*, *definitions* and *queries* is described as follows:

$$e ::= c \mid x \mid (g\ e_1 \ldots e_n) \mid (\mathsf{D}\ e_1 \ldots e_n) \mid (\texttt{if}\ e_1\ e_2\ e_3) \mid (\texttt{lambda}\ (x_1 \ldots x_n)\ e) \mid (e_1\ e_2 \ldots e_n)$$

$$\texttt{d} ::= (\texttt{define}\ x\ e)$$

$$\texttt{q} ::= (\texttt{query}\ d_1 \ldots d_n\ e\ e_{cond})$$

To make the translation more intuitive, it is convenient to add to the target language a let-expression of the form $\texttt{let}\ x = M\ \texttt{in}\ N$, that can be interpreted as syntactic sugar for $(\lambda x.N)\ M$, and sequencing $M; N$ that stands for $\lambda\star.N\ M$ where $\star$ as usual stands for a variable that does not appear free in any of the terms under consideration.

The rules for translating CHURCH expressions to the calculus are shown in Figure 2, where $fv(e)$ denotes the set of free variables in expression e and *fix* $x.M$ is a call-by-value fixpoint combinator $\lambda y.N_{fix}N_{fix}(\lambda x.M)y$ where N_{fix} is $\lambda z.\lambda w.w(\lambda y.((zz)w)y)$. Observe that $(\mathit{fix}\ x.M)V$ evaluates to $M\{(\mathit{fix}\ x.M)/x\}V$ deterministically. We assume that for each distribution identifier D of arity k, there is a deterministic function $\mathrm{pdf}_{\mathsf{D}}$ of arity $k + 1$ that calculates the corresponding density at the given point.

$$\frac{G \in \mathcal{GV}}{G \Downarrow_1^{[]} G}\ (\textsc{Eval Val}) \qquad \frac{w = \mathrm{pdf}_{\mathsf{D}}(\vec{c}, c) \qquad w > 0}{\mathsf{D}(\vec{c}) \Downarrow_w^{[c]} c}\ (\textsc{Eval Random})$$

$$\frac{\mathrm{pdf}_{\mathsf{D}}(\vec{c}, c) = 0}{\mathsf{D}(\vec{c}) \Downarrow_0^{[c]} \texttt{fail}}\ (\textsc{Eval Random Fail}) \qquad \frac{}{g(\vec{c}) \Downarrow_1^{[]} \sigma_g(\vec{c})}\ (\textsc{Eval Prim})$$

$$\frac{M \Downarrow_{w_1}^{s_1} \lambda x.P \qquad N \Downarrow_{w_2}^{s_2} V \qquad P[V/x] \Downarrow_{w_3}^{s_3} G}{M\ N \Downarrow_{w_1 \cdot w_2 \cdot w_3}^{s_1 @ s_2 @ s_3} G}\ (\textsc{Eval Appl}) \qquad \frac{M \Downarrow_w^s \texttt{fail}}{M\ N \Downarrow_w^s \texttt{fail}}\ (\textsc{Eval Appl Raise1})$$

$$\frac{M \Downarrow_w^s c}{M\ N \Downarrow_w^s \texttt{fail}}\ (\textsc{Eval Appl Raise2}) \qquad \frac{M \Downarrow_{w_1}^{s_1} \lambda x.P \qquad N \Downarrow_{w_2}^{s_2} \texttt{fail}}{M\ N \Downarrow_{w_1 \cdot w_2}^{s_1 @ s_2} \texttt{fail}}\ (\textsc{Eval Appl Raise3})$$

$$\frac{M \Downarrow_w^s G}{\texttt{if true then } M \texttt{ else } N \Downarrow_w^s G}\ (\textsc{Eval If True}) \qquad \frac{N \Downarrow_w^s G}{\texttt{if false then } M \texttt{ else } N \Downarrow_w^s G}\ (\textsc{Eval If False})$$

$$\frac{c \in (0, 1]}{\texttt{score}(c) \Downarrow_c^{[]} \texttt{true}}\ (\textsc{Eval Score}) \qquad \frac{T \text{ is an erroneous redex}}{T \Downarrow_1^{[]} \texttt{fail}}\ (\textsc{Eval Fail})$$

Figure 1. Sampling-Based Big Step Semantics

$\langle c \rangle_e = c$
$\langle x \rangle_e = x$
$\langle g\ e_1, \ldots, e_n \rangle_e =$
  $\texttt{let}\ x_1 = e_1\ \texttt{in}\ \ldots \texttt{let}\ x_n = e_n\ \texttt{in}\ g(x_1, \ldots, x_n)$
  where $x_1, \ldots, x_n \notin fv(e_1) \cup \cdots \cup fv(e_n)$
$\langle \mathsf{D}\ e_1, \ldots e_n \rangle_e =$
  $\texttt{let}\ x_1 = e_1\ \texttt{in}\ \ldots \texttt{let}\ x_n = e_n\ \texttt{in}\ \mathsf{D}(x_1, \ldots, x_n)$
  where $x_1, \ldots, x_n \notin fv(e_1) \cup \cdots \cup fv(e_n)$
$\langle \texttt{lambda}\ ()\ e \rangle_e = \lambda x.\langle e \rangle_e$ where $x \notin fv(e)$
$\langle \texttt{lambda}\ x\ e \rangle_e = \lambda x.\langle e \rangle_e$
$\langle \texttt{lambda}\ (x_1\ \ldots\ x_n)\ e \rangle_e = \lambda x_1.\langle \texttt{lambda}\ (x_2\ \ldots\ x_n)\ e \rangle_e$
$\langle e_1\ e_2 \rangle_e = \langle e_1 \rangle_e\ \langle e_2 \rangle_e$
$\langle e_1\ e_2\ \ldots\ e_n \rangle_e = \langle (e_1\ e_2)\ \ldots\ e_n \rangle_e$
$\langle \texttt{if}\ e_1\ e_2\ e_3 \rangle_e = \texttt{let}\ x = e_1\ \texttt{in}\ (\texttt{if}\ x\ \texttt{then}\ \langle e_2 \rangle_e\ \texttt{else}\ \langle e_3 \rangle_e)$
  where $x \notin fv(e_2) \cup fv(e_3)$

$\langle \texttt{query}\ (\texttt{define}\ x_1\ e_1) \ldots (\texttt{define}\ x_n\ e_n)\ e_{out}\ e_{cond} \rangle =$
  $\texttt{let}\ x_1 = (\mathit{fix}\ x_1.\langle e_1 \rangle_e)\ \texttt{in}$
  $\ldots$
  $\texttt{let}\ x_n = (\mathit{fix}\ x_n.\langle e_n \rangle_e)\ \texttt{in}$
  $\texttt{let}\ b = e_{cond}\ \texttt{in}$
  $\texttt{if}\ b\ \texttt{then}\ e_{out}\ \texttt{else fail}$

Figure 2. Translation of CHURCH

In addition to expressions presented here, CHURCH also supports *stochastic memoization* (Goodman et al. 2008) by means of a `mem` function, which, applied to any given function, produces a version of it that always returns the same value when applied to the same arguments. This feature allows for functions of integers to be treated as infinite lazy lists of random values, and is useful in defining some nonparametric models, such as the Dirichlet Process.

It would be straightforward to add support for memoization in our encoding by changing the translation to state-passing style, but we omit this standard extension for the sake of brevity.

2.4 Example: Geometric Distribution

To illustrate the sampling-based semantics, recall the geometric distribution example from Section 1. It translates to the following program in the core calculus:

$\texttt{let}\ \mathit{flip} = \lambda x.(\mathsf{rnd}() < x)\ \texttt{in}$
$\texttt{let}\ \mathit{geometric} =$
  $(\mathit{fix}\ g.$
    $\lambda p.\ (\texttt{let}\ y = \mathsf{rnd}() < p\ \texttt{in}$
      $\texttt{if}\ y\ \texttt{then}\ 0\ \texttt{else}\ 1 + (g\ p)))\ \texttt{in}$
$\texttt{let}\ n = \mathit{fix}\ n'.\mathit{geometric}\ 0.5\ \texttt{in}$
$\texttt{let}\ b = n > 1\ \texttt{in}$
$\texttt{if}\ b\ \texttt{then}\ n\ \texttt{else fail}$

Suppose we want to evaluate this program on the random trace $s = [0.7, 0.8, 0.3]$. By (EVAL APPL), we can substitute the definitions of *flip* and *geometric* in the remainder of the program, without consuming any elements of the trace nor changing the weight of the sample. Then we need to evaluate *geometric* 0.5.

It can be shown (by repeatedly applying (EVAL APPL)) that for any lambda-abstraction $\lambda x.M$, $M\{(\mathit{fix}\ x.M)/x\}\ V \Downarrow_w^s G$ if and only if $(\mathit{fix}\ x.M)\ V \Downarrow_w^s G$, which allows us to unfold the recursion. Applying the unfolded definition of *geometric* to the argument 0.5 yields an expression of the form

$\texttt{let}\ y = \mathsf{rnd}() < 0.5\ \texttt{in}$
  $\texttt{if}\ y\ \texttt{then}\ 0\ \texttt{else}\ 1 + (\ldots).$

For the first random draw, we have $\mathsf{rnd}() \Downarrow_1^{[0.7]} 0.7$ by (EVAL RANDOM) (because the density of rnd is 1 on the interval $[0, 1]$) and so (EVAL PRIM) gives $\mathsf{rnd}() < 0.5 \Downarrow_1^{[0.7]} \texttt{false}$. After unfolding the recursion two more times, evaluating the subsequent "flips" yields $\mathsf{rnd}() < 0.5 \Downarrow_1^{[0.8]} \texttt{false}$ and $\mathsf{rnd}() < 0.5 \Downarrow_1^{[0.3]} \texttt{true}$. By (EVAL IF TRUE), the last if-statement evaluates to 0, terminating the recursion. Combining the results by (EVAL APPL), (EVAL IF FALSE) and (EVAL PRIM), we arrive at $\mathit{geometric}\ 0.5 \Downarrow_1^{[0.7,0.8,0.3]} 2$.

At this point, it is straightforward to see that the condition in the if-statement on the final line is satisfied, and hence the program reduces with the given trace to the value 2 with weight 1.

This program actually yields weight 1 for every trace that returns an integer value. This may seem counter-intuitive, because

clearly not all outcomes have the same probability. However, the probability of a given outcome is given by an integral over the space of traces, as described in Section 3.4.

2.5 Soft Constraints and `score`

The geometric distribution example in Section 2.4 uses a *hard constraint*: program execution fails and the value of n is discarded whenever the Boolean predicate $n > 1$ is not satisfied. In many machine learning applications we want to use a different kind of constraint that models noisy data. For instance, if c is the known output of a sensor that shows an approximate value of some unknown quantity x, we want to assign higher probabilities to values of x that are closer to c. This is sometimes known as a *soft constraint*.

One naive way to implement a soft constraint is to use a hard constraint with a success probability based on $|x - c|$, for instance,

$$\mathit{condition}\ x\ c\ M := \texttt{if}\ \mathit{flip}(\mathit{exp}(-(x-c)^2))\ \texttt{then}\ M\ \texttt{else}\ \texttt{fail}.$$

Then $\mathit{condition}\ x\ c\ M$ has the effect of continuing as M with probability $\mathit{exp}(-(x-c)^2)$, and otherwise terminating execution. In the context of a sampling-based semantics, it has the effect of adding a uniform sample from $[0, \mathit{exp}(-(x-c)^2))$ to any successful trace, in addition to introducing more failing traces.

Instead, our calculus includes a primitive `score`, that avoids both adding dummy samples and introducing more failing traces. It also admits the possibility of using efficient gradient-based methods of inference (e.g., Homan and Gelman (2014)). Using `score`, the above conditioning operator can be redefined as

$$\mathit{score\text{-}condition}\ x\ c\ M := \texttt{score}(\mathit{exp}(-(x-c)^2)); M$$

2.6 Example: Linear Regression

For an example of soft constraints, consider the ubiquitous linear regression model $y = m \cdot x + b + \mathit{noise}$, where x is often a known feature and y an observable outcome variable. We can model the noise as drawn from a Gaussian distribution with mean 0 and variance 1/2 by letting the success probability be given by the function `squash` below.

The following query[2] predicts the y-coordinate for $x = 4$, given observations of four points: $(0, 0)$, $(1, 1)$, $(2, 4)$, and $(3, 6)$. (We use the abbreviation `(define (`$f\ x_1\ \ldots\ x_n$`) `e`)` for `(define `f `(lambda (`$x_1\ \ldots\ x_n$`) `e`)`, and use `and` for multiadic conjunction.)

```
(query
   (define (sqr x) (* x x)))
   (define (squash x y) (exp(- (sqr(- x y)))))
   (define (flip p) (< (rnd) p))
   (define (softeq x y) (flip (squash x y))))

   (define m (gaussian 0 2))
   (define b (gaussian 0 2))
   (define (f x) (+ (* m x) b))

   (f 4)  ;; predict y for x=4

   (and (softeq (f 0) 0) (softeq (f 1) 1)
        (softeq (f 2) 4) (softeq (f 3) 6))
```

The model described above puts independent Gaussian priors on m and b. The condition of the query states that all observed ys are (soft) equal to $k \cdot x + m$. Assuming that `softeq` is used only to define constraints (i.e., positively), we can avoid the nuisance parameter that arises from each `flip` by redefining `softeq` as follows (given a `score` primitive in CHURCH, mapped to `score`$(-)$ in our λ-calculus):

[2] Cf. `http://forestdb.org/models/linear-regression.html`.

$$\begin{aligned}
E[g(\vec{c})] &\xrightarrow{\text{det}} E[\sigma_g(\vec{c})] \\
E[(\lambda x.M)\ V] &\xrightarrow{\text{det}} E[M\{V/x\}] \\
E[\texttt{if 1 then } M_2 \texttt{ else } M_3] &\xrightarrow{\text{det}} E[M_2] \\
E[\texttt{if 0 then } M_2 \texttt{ else } M_3] &\xrightarrow{\text{det}} E[M_3] \\
E[T] &\xrightarrow{\text{det}} E[\texttt{fail}] \\
E[\texttt{fail}] &\xrightarrow{\text{det}} \texttt{fail} \quad \text{if } E \text{ is not } [\cdot]
\end{aligned}$$

Figure 3. Deterministic Reduction.

```
(define (softeq x y) (score (squash x y)))
```

3. Sampling-Based Operational Semantics

In this section, we further investigate sampling-based semantics for our calculus. First, we introduce *small*-step sampling-based semantics and prove it equivalent to its big-step sibling as introduced in Section 2.2. Then, we associate to any closed term M two sub-probability distributions: one on the set of random traces, and the other on the set of return values. This requires some measure theory, recalled in Section 3.2.

3.1 Small-step Sampling-based Semantics

We define small-step call-by-value evaluation. *Evaluation contexts* are defined as follows:

$$E ::= [\cdot] \mid EM \mid (\lambda x.M)E$$

We let $\mathcal{C}$ be the set of all closed evaluation contexts, i.e., where every occurrence of a variable x is as a subterm of $\lambda x.M$. The term obtained by replacing the only occurrence of $[\cdot]$ in E by M is indicated as $E[M]$. *Redexes* are generated by the following grammar:

$$\begin{aligned}
R ::= {} & (\lambda x.M)V \mid \mathsf{D}(\vec{c}) \mid g(\vec{c}) \mid \texttt{score}(c) \\
& \mid \texttt{fail} \mid \texttt{if true then } M \texttt{ else } N \\
& \mid \texttt{if false then } M \texttt{ else } N \mid T
\end{aligned}$$

Reducible terms are those closed terms M that can be written as $E[R]$.

LEMMA 1. *For every closed term M, either M is a generalized value or there are unique E, R such that $M = E[R]$. Moreover, if M is not a generalized value and $R = \texttt{fail}$, then E is proper, that is, $E \neq [\cdot]$.*

PROOF. This is an easy induction on the structure of M. □

Deterministic reduction is the relation $\xrightarrow{\text{det}}$ on closed terms defined in Figure 3. Rules of small-step reduction are given in Figure 4. We let *multi-step reduction* be the inductively defined relation $(M, w, s) \Rightarrow (M', w', s')$ if and only if $(M, w, s) = (M', w', s')$ or $(M, w, s) \to (M'', w'', s'') \Rightarrow (M', w', s')$ for some M'', w'', s''. As can be easily verified, the multi-step reduction of a term to a generalized value is deterministic once the underlying trace and weight are kept fixed:

LEMMA 2. *If both $(M, w, s) \Rightarrow (G', w', s')$ and $(M, w, s) \Rightarrow (G'', w'', s'')$, then $G' = G''$, $w' = w''$ and $s' = s''$.*

$$\frac{M \xrightarrow{\text{det}} N}{(M, w, s) \to (N, w, s)}\ (\textsc{Red Pure}) \qquad \frac{c \in (0, 1]}{(E[\texttt{score}(c)], w, s) \to (E[\texttt{true}], c \cdot w, s)}\ (\textsc{Red Score})$$

$$\frac{w' = \mathrm{pdf}_{\mathsf{D}}(\vec{c}, c) \qquad w' > 0}{(E[\mathsf{D}(\vec{c})], w, c :: s) \to (E[c], w \cdot w', s)}\ (\textsc{Red Random}) \qquad \frac{\mathrm{pdf}_{\mathsf{D}}(\vec{c}, c) = 0}{(E[\mathsf{D}(\vec{c})], w, c :: s) \to (E[\texttt{fail}], 0, s)}\ (\textsc{Red Random Fail})$$

Figure 4. Small-step sampling-based operational semantics

Reduction can take place in any evaluation context, provided the result is not a failure. Moreover, multi-step reduction is a transitive relation. This is captured by the following lemmas.

LEMMA 3. *For any E, if $(M, w, s) \Rightarrow (M', w', s')$ and $M' \neq \texttt{fail}$, then we have $(E[M], w, s) \Rightarrow (E[M'], w', s')$.*

LEMMA 4. *If both $(M, 1, s) \Rightarrow (M', w', [])$ and $(M', 1, s') \Rightarrow (M'', w'', [])$, then $(M, 1, s@s') \Rightarrow (M'', w' \cdot w'', [])$.*

The following directly relates the small-step and big-step semantics, saying that the latter is invariant on the former:

LEMMA 5. *If $(M, 1, s) \to (M', w, [])$ and $M' \Downarrow^{s'}_{w'} G$, then $M \Downarrow^{s@s'}_{w \cdot w'} G$.*

Finally, we have all the ingredients to show that the small-step and the big-step sampling-based semantics both compute the same traces with the same weights.

THEOREM 1. *$M \Downarrow^{s}_{w} G$ if and only if $(M, 1, s) \Rightarrow (G, w, [])$.*

PROOF. The left to right implication is an induction on the derivation of $M \Downarrow^{s}_{w} G$. The right to left implication can be proved by an induction on the length of the derivation of $(M, 1, s) \Rightarrow (G, w, [])$, with appeal to Lemma 5. □

As a corollary of Theorem 1 and Lemma 2 we obtain:

LEMMA 6. *If $M \Downarrow^{s}_{w} G$ and $M \Downarrow^{s}_{w'} G'$ then $w = w'$ and $G = G'$.*

At this point, we have defined intuitive operational semantics based on the consumption of an explicit trace of randomness, but we have defined no distributions. In the rest of this section we show that this semantics indeed associates a sub-probability distribution with each term. Before proceeding, however, we need some measure theory.

3.2 Some Measure-Theoretic Preliminaries.

We begin by recapitulating some standard definitions for sub-probability distributions and kernels over metric spaces. For a more complete, tutorial-style introduction to measure theory, see Billingsley (1995), Panangaden (2009), or another standard textbook or lecture notes.

A σ-algebra (over a set X) is a set Σ of subsets of X that contains $\varnothing$, and is closed under complement and countable union (and hence is closed under countable intersection). Let the σ-algebra *generated* by S, written $\sigma(S)$, be the least σ-algebra over $\cup S$ that is a superset of S.

We write $\mathbb{R}_+$ for $[0, \infty]$ and $\mathbb{R}_{[0,1]}$ for the interval $[0, 1]$. A *metric space* is a set X with a symmetric *distance function* $\delta : X \times X \to \mathbb{R}_+$ that satisfies the triangle inequality $\delta(x, z) \leq \delta(x, y) + \delta(y, z)$ and the axiom $\delta(x, x) = 0$. We write $\mathbf{B}(x, r) \triangleq \{y \mid \delta(x, y) < r\}$ for the open ball around x of radius r. We equip $\mathbb{R}_+$ and $\mathbb{R}_{[0,1]}$ with the standard metric $\delta(x, y) = |x - y|$, and products of metric spaces with the Manhattan metric (e.g., $\delta((x_1, x_2), (y_1, y_2)) = \delta(x_1, y_1) + \delta(x_2, y_2)$). The *Borel σ-algebra* on a metric space (X, δ) is $\mathcal{B}(X, \delta) \triangleq \sigma(\{\mathbf{B}(x, r) \mid x \in X \wedge r > 0\})$. We often omit the arguments to $\mathcal{B}$ when they are clear from the context.

A *measurable space* is a pair (X, Σ) where X is a set of possible outcomes, and $\Sigma \subseteq \mathcal{P}(X)$ is a σ-algebra of *measurable sets*. As an example, consider the extended positive real numbers $\mathbb{R}_+$ equipped with the Borel σ-algebra $\mathcal{R}$, i.e. the set $\sigma(\{(a, b) \mid a, b \geq 0\})$ which is the smallest σ-algebra containing all open (and closed) intervals. We can create finite products of measurable spaces by iterating the construction $(X, \Sigma) \times (X', \Sigma') = (X \times X', \sigma(A \times B \mid A \in \Sigma \wedge B \in \Sigma'))$. If (X, Σ) and (X', Σ') are measurable spaces, then the function $f : X \to X'$ is *measurable* if and only if for all $A \in \Sigma'$, $f^{-1}(A) \in \Sigma$, where the *inverse image* $f^{-1} : \mathcal{P}(X') \to \mathcal{P}(X)$ is given by $f^{-1}(A) \triangleq \{x \in X \mid f(x) \in A\}$.

A *measure* μ on (X, Σ) is a function from Σ to $\mathbb{R}_+$, that is (1) zero on the empty set, that is, $\mu(\varnothing) = 0$, and (2) countably additive, that is, $\mu(\cup_i A_i) = \Sigma_i \mu(A_i)$ if $A_1, A_2, \ldots$ are pair-wise disjoint. The measure μ is called a *(sub-probability) distribution* if $\mu(X) \leq 1$ and *finite* if $\mu(X) \neq \infty$. If μ, ν are finite measures and $c \geq 0$, we write $c \cdot \mu$ for the finite measure $A \mapsto c \cdot (\mu(A))$ and $\mu + \nu$ for the finite measure $A \mapsto \mu(A) + \nu(A)$. We write $\mathbf{0}$ for the zero measure $A \mapsto 0$. For any element x of X, the Dirac measure $\delta(x)$ is defined as follows:

$$\delta(x)(A) = \begin{cases} 1 & \text{if } x \in A; \\ 0 & \text{otherwise.} \end{cases}$$

A *measure space* is a triple $\mathscr{M} = (X, \Sigma, \mu)$ where μ is a measure on the measurable space (X, Σ). Given a measurable function $f : X \to \mathbb{R}_+$, the *integral* of f over $\mathscr{M}$ can be defined following Lebesgue's theory and denoted as either of

$$\int f \, d\mu = \int f(x) \, \mu(dx) \in \mathbb{R}_+.$$

The Iverson brackets $[P]$ are 1 if predicate P is true, and 0 otherwise. We then write

$$\int_A f \, d\mu \triangleq \int f(x) \cdot [x \in A] \, \mu(dx).$$

We equip some measurable spaces (X, Σ) with a *stock measure* μ. We then write $\int f(s) \, ds$ (or shorter, $\int f$) for $\int f \, d\mu$ when f is measurable $f : X \to \mathbb{R}_+$. In particular, we let the stock measure on $(\mathbb{R}^n, \mathcal{B})$ be the Lebesgue measure λ_n.

A function f is a *density* of a measure ν (with respect to the measure μ) if $\nu(A) = \int_A f \, d\mu$ for all measurable A.

Given a measurable set A from (X, Σ), we write $\Sigma|_A$ for the *restriction* of Σ to elements *in* A, *i.e.*, $\Sigma|_A = \{B \cap A \mid B \in \Sigma\}$. Then $(A, \Sigma|_A)$ is a measurable space. Any distribution μ on (X, Σ) trivially yields a distribution $\mu|_A$ on $(A, \Sigma|_A)$ by $\mu|_A(B) = \mu(B)$.

3.3 Measure Space of Program Traces

In this section, we construct a measure space on the set $\mathbb{S}$ of program traces: (1) we define a measurable space $(\mathbb{S}, \mathcal{S})$ and (2)

we equip it with a stock measure μ to obtain our measure space $(\mathbb{S}, \mathcal{S}, \mu)$.

The Measurable Space of Program Traces To define the semantics of a program as a measure on the space of random choices, we first need to define a measurable space of program traces. Since a program trace is a sequence of real numbers of an arbitrary length (possibly 0), the set of all program traces is $\mathbb{S} = \biguplus_{n\in\mathbb{N}} \mathbb{R}^n$. Now, let us define the σ-algebra $\mathcal{S}$ on $\mathbb{S}$ as follows: let $\mathcal{B}_n$ be the Borel σ-algebra on $\mathbb{R}^n$ (we take $\mathcal{B}_0$ to be $\{\{[]\}, \{\}\}$). Consider the class of sets $\mathcal{S}$ of the form:

$$A = \biguplus_{n\in\mathbb{N}} H_n$$

where $H_n \in \mathcal{B}_n$ for all n. Then $\mathcal{S}$ is a σ-algebra, and so $(\mathbb{S}, \mathcal{S})$ is a measurable space.

LEMMA 7. *$\mathcal{S}$ is a σ-algebra on $\mathbb{S}$.*

Stock Measure on Program Traces Since each primitive distribution D has a density, the probability of each random value (and thus of each trace of random values) is zero. Instead, we define the trace and transition probabilities in terms of densities, with respect to the stock measure μ on $(\mathbb{S}, \mathcal{S})$ defined below,

$$\mu\left(\biguplus_{n\in\mathbb{N}} H_n\right) = \sum_{n\in\mathbb{N}} \lambda_n(H_n)$$

where $\lambda_0 = \delta([])$ and λ_n is the Lebesgue measure on $\mathbb{R}^n$ for $n > 0$.

LEMMA 8. *μ is a measure on $(\mathbb{S}, \mathcal{S})$.*

3.4 Distributions $\langle\!\langle M \rangle\!\rangle$ and $[\![M]\!]_{\mathbb{S}}$ Given by Sampling-Based Semantics

The result of a closed term M on a given trace is

$$\mathbf{O}_M(s) = \begin{cases} G & \text{if } M \Downarrow_w^s G \text{ for some } w \in \mathbb{R}_+ \\ \texttt{fail} & \text{otherwise.} \end{cases}$$

The density of termination of a closed term M on a given trace is defined as follows.

$$\mathbf{P}_M(s) = \begin{cases} w & \text{if } M \Downarrow_w^s G \text{ for some } G \in \mathcal{GV} \\ 0 & \text{otherwise} \end{cases}$$

This density function induces a distribution $\langle\!\langle M \rangle\!\rangle$ on traces defined as $\langle\!\langle M \rangle\!\rangle(A) := \int_A \mathbf{P}_M$.

By inverting the result function $\mathbf{O}_M$, we also obtain a distribution $[\![M]\!]_{\mathbb{S}}$ over generalised values (also called a *result distribution*). It can be computed by integrating the density of termination over all traces that yield the generalised values of interest.

$$[\![M]\!]_{\mathbb{S}}(A) := \langle\!\langle M \rangle\!\rangle(\mathbf{O}_M^{-1}(A)) = \int \mathbf{P}_M(s) \cdot [\mathbf{O}_M(s) \in A]\, ds.$$

As an example, for the geometric distribution example of Section 2.4 we have $\mathbf{O}_{geometric\ 0.5}(s) = n$ if $s \in [0.5, 1]^n[0, 0.5)$, and otherwise $\mathbf{O}_{geometric\ 0.5}(s) = \texttt{fail}$. Similarly, we have $\mathbf{P}_{geometric\ 0.5}(s) = 1$ if $s \in [0.5, 1]^n[0, 0.5)$ for some n, and otherwise 0. We then obtain

$$\langle\!\langle geometric\ 0.5 \rangle\!\rangle(A) = \sum_{n\in\mathbb{N}} \lambda_{n+1}(A \cap \{[0.5, 1]^n[0, 0.5)\}) \text{ and}$$

$$[\![geometric\ 0.5]\!]_{\mathbb{S}}(\{n\}) = \int [s \in \{[0.5, 1]^n[0, 0.5)\}]\, ds = \frac{1}{2^{n+1}}.$$

As seen above, we use the exception `fail` to model the failure of a hard constraint. To restrict attention to normal termination, we modify $\mathbf{P}_M$ as follows.

$$\mathbf{P}_M^{\mathcal{V}}(s) = \begin{cases} w & \text{if } M \Downarrow_w^s V \text{ for some } V \in \mathcal{V} \\ 0 & \text{otherwise.} \end{cases}$$

$$\begin{aligned}
d(x, x) &= 0 \\
d(c, d) &= |c - d| \\
d(MN, LP) &= d(M, L) + d(N, P) \\
d(g(V_1, \ldots, V_n), &g(W_1, \ldots, W_n)) \\
&= d(V_1, W_1) + \cdots + d(V_n, W_n) \\
d(\lambda x.M, \lambda x.N) &= d(M, N) \\
d(\mathsf{D}(V_1, \ldots, V_n), &\mathsf{D}(W_1, \ldots, W_n)) \\
&= d(V_1, W_1) + \cdots + d(V_n, W_n) \\
d(\texttt{score}(V), \texttt{score}(W)) &= d(V, W) \\
d(\texttt{if } V \texttt{ then } M \texttt{ else } N, &\texttt{if } W \texttt{ then } L \texttt{ else } P) \\
&= d(V, W) + d(M, L) + d(N, P) \\
d(\texttt{fail}, \texttt{fail}) &= 0 \\
d(M, N) &= \infty \text{ otherwise}
\end{aligned}$$

Figure 5. Metric d on terms.

As above, this density function generates distributions over traces and values as, respectively

$$\langle\!\langle M \rangle\!\rangle^{\mathcal{V}}(A) := \int_A \mathbf{P}_M^{\mathcal{V}} = \langle\!\langle M \rangle\!\rangle(A \cap \mathbf{O}_M^{-1}(\mathcal{V}))$$

$$([\![M]\!]_{\mathbb{S}})|_{\mathcal{V}}(A) = [\![M]\!]_{\mathbb{S}}(A \cap \mathcal{V}) = \int \mathbf{P}_M^{\mathcal{V}}(s) \cdot [\mathbf{O}_M(s) \in A]\, ds$$

To show that the above definitions make sense measure-theoretically, we first define the measurable space of terms $(\Lambda, \mathcal{M})$, where $\mathcal{M}$ is the set of Borel-measurable sets of terms with respect to the recursively defined metric d in Figure 5.

LEMMA 9. *For any closed term M, the functions $\mathbf{P}_M$, $\mathbf{O}_M$ and $\mathbf{P}_M^{\mathcal{V}}$ are all measurable; $\langle\!\langle M \rangle\!\rangle$ and $\langle\!\langle M \rangle\!\rangle^{\mathcal{V}}$ are measures on $(\mathbb{S}, \mathcal{S})$; $[\![M]\!]_{\mathbb{S}}$ is a measure on $(\mathcal{GV}, \mathcal{M}|_{\mathcal{GV}})$; and $([\![M]\!]_{\mathbb{S}})|_{\mathcal{V}}$ is a measure on $(\mathcal{V}, \mathcal{M}|_{\mathcal{V}})$.*

4. Distribution-Based Operational Semantics

In this section we introduce small- and big-step distribution-based operational semantics, where the small-step semantics is a generalisation of Jones (1990) to continuous distributions. We prove correspondence between the semantics using some non-obvious properties of kernels. Moreover, we will prove that the distribution-based semantics are equivalent to the sampling-based semantics from Section 3. A term will correspond to a distribution over generalised values, below called a result distribution.

4.1 Sub-Probability Kernels

If (X, Σ) and (Y, Σ') are measurable spaces, then a function $Q : X \times \Sigma' \to \mathbb{R}_{[0,1]}$ is called a *(sub-probability) kernel* (from (X, Σ) to (Y, Σ')) if

1. for every $x \in X$, $Q(x, \cdot)$ is a sub-probability distribution on (Y, Σ'); and
2. for every $A \in \Sigma'$, $Q(\cdot, A)$ is a non-negative measurable function $X \to \mathbb{R}_{[0,1]}$.

The measurable function $q : X \times Y \to \mathbb{R}_+$ is said to be a *density* of kernel Q with respect to a measure μ on (Y, Σ') if $Q(v, A) = \int_A q(v, y)\, \mu(dy)$ for all $v \in X$ and $A \in \Sigma'$. When Q is a kernel, note that $\int f(y)\, Q(x, dy)$ denotes the integral of f with respect to the measure $Q(x, \cdot)$.

Kernels can be composed in the following ways: If Q_1 is a kernel from (X_1, Σ_1) to (X_2, Σ_2) and Q_2 is a kernel from (X_2, Σ_2)

$$\frac{n > 0}{G \Rightarrow_n \delta(G)}\ (\text{DRED VAL}) \qquad \frac{}{M \Rightarrow_0 \mathbf{0}}\ (\text{DRED EMPTY})$$

$$\frac{M \to \mathscr{D} \qquad \{N \Rightarrow_n \mathscr{E}_N\}_{N \in \mathrm{supp}(\mathscr{D})}}{M \Rightarrow_{n+1} A \mapsto \int \mathscr{E}_N(A)\, \mathscr{D}(dN)}\ (\text{DRED STEP})$$

Figure 6. Step-Indexed Approximation Small-Step Semantics.

to (X_3, Σ_3), then $Q_2 \circ Q_1 : (x, A) \mapsto \int Q_2(y, A)\, Q_1(x, dy)$ is a kernel from (X_1, Σ_1) to (X_3, Σ_3). Moreover, if Q_1 is a kernel from (X_1, Σ_1) to (X_2, Σ_2) and Q_2 is a kernel from (X_1', Σ_1') to (X_2', Σ_2'), then $Q_1 \times Q_2 : ((x, y), (A \times B)) \mapsto Q_1(x, A) \cdot Q_2(y, B)$ uniquely extends to a kernel from $(X_1, \Sigma_1) \times (X_1', \Sigma_1')$ to $(X_2, \Sigma_2) \times (X_2', \Sigma_2')$.

4.2 Approximation Small-Step Semantics

The first thing we need to do is to generalize deterministic reduction into a relation between closed terms and term *distributions*. If μ is a measure on terms and E is an evaluation context, we let $E\{\mu\}$ be the push-forward measure $A \mapsto \mu(\{M \mid E[M] \in A\})$.

One-step evaluation is a relation $M \to \mathscr{D}$ between closed terms M and distributions $\mathscr{D}$ on terms, defined as follows:

$$E[\mathsf{D}(\vec{c})] \to E\{\mu_{\mathsf{D}(\vec{c})}\}$$
$$E[M] \to \delta(E[N]) \text{ if } M \xrightarrow{\text{det}} N$$
$$E[\mathtt{score}(c)] \to c \cdot \delta(E[\mathtt{true}]) \text{ if } 0 < c \leq 1$$

We first of all want to show that one-step reduction is essentially deterministic, and that we have a form of deadlock-freedom.

LEMMA 10. *For every closed term M, either M is a generalized value or there is a unique $\mathscr{D}$ such that $M \to \mathscr{D}$.*

We need to prove the just introduced notion of one-step reduction to support composition. This is captured by the following result.

LEMMA 11. *$\to$ is a sub-probability kernel.*

PROOF. Lemma 10 tells us that $\to$ can be seen as a function $\hat{\to}$. The fact that $\hat{\to}(M, \cdot)$ is a distribution is easily verified. On other hand, the fact that $\hat{\to}(\cdot, A)$ is measurable amounts to proving that $(\hat{\to}(\cdot, A))^{-1}(B)$ is a measurable set of terms whenever B is a measurable set of real numbers. Proving that requires some care, and a proof can be found in (Borgström et al. 2015). □

Given a family $\{\mathscr{D}_M\}_{M \in A}$ of distributions indexed by terms in a measurable set A of terms, and a measurable set B, we often write, with an abuse of notation, $\mathscr{D}_M(B)$ for the function that assigns to any term $M \in A$ the real number $\mathscr{D}_M(B)$. The *step-indexed approximation small-step semantics* is the family of n-indexed relations $M \Rightarrow_n \mathscr{D}$ between terms and result distributions inductively defined in Figure 6. Since generalised values have no transitions (there is no $\mathscr{D}$ such that $G \to \mathscr{D}$), the rules above are disjoint and so there is at most one $\mathscr{D}$ such that $M \to_n \mathscr{D}$. Compared to the discrete case (Jones 1990), the step-index n is needed to ensure that the integral in (DRED STEP) is defined.

LEMMA 12. *For every $n \in \mathbb{N}$, the function $\Rightarrow_n$ is a kernel.*

PROOF. This is an induction on n, exploiting Lemma 11 and the fact that sub-probability kernels compose. □

LEMMA 13. *For every closed term M and for every $n \in \mathbb{N}$ there is a unique distribution $\mathscr{D}$ such that $M \Rightarrow_n \mathscr{D}$.*

PROOF. This is an easy consequence of Lemma 12. □

4.3 Approximation Big-Step Semantics

The *step-indexed approximation big-step semantics* $M \Downarrow_n \mathscr{D}$ is the n-indexed family of relations between terms and result distributions inductively defined by the rules in Figure 7.

Above, the rule for applications is the most complex, with the resulting distribution consisting of three exceptional terms in addition to the normal case. To better understand this rule, one can study what happens if we replace general applications with a let construct plus application of values to values. Then we would end up having the following three rules, instead of the rule for application above:

$$\frac{M \Downarrow_n \mathscr{D} \qquad \{N\{V/x\} \Downarrow_n \mathscr{E}_V\}_{V \in \mathrm{supp}(\mathscr{D})}}{\mathtt{let}\ x = M\ \mathtt{in}\ N \Downarrow_{n+1} A \mapsto \begin{pmatrix} \mathscr{D}|_{\{\mathtt{fail}\}}(A) \\ +\mathscr{D}(\mathbb{R}) \cdot [\mathtt{fail} \in A] \\ +\int \mathscr{E}_V(A)\, \mathscr{D}|_{\mathcal{V}}(dV) \end{pmatrix}}$$

$$\frac{M\{V/x\} \Downarrow_n \mathscr{E}}{(\lambda x.M)V \Downarrow_{n+1} \mathscr{E}} \qquad \frac{n > 0}{c\ V \Downarrow_n \delta(\mathtt{fail})}$$

The existence of the integral in rule (DEVAL APPL) is guaranteed by a lemma analogous to Lemma 12.

LEMMA 14. *For every $n \in \mathbb{N}$, the function $\Downarrow_n$ is a kernel.*

This can be proved by induction on n, with the most difficult case being precisely the one of applications. Composition and product properties of kernels are the key ingredients there.

4.4 Beyond Approximations

The set of result distributions with the pointwise order forms an $\omega\mathbf{CPO}$, and thus any denumerable, directed set of result distributions has a least upper bound. One can define the *small-step semantics* and the *big-step semantics* as, respectively, the two distributions

$$[\![M]\!]_{\Rightarrow} = \sup\{\mathscr{D} \mid M \to_n \mathscr{D}\}$$
$$[\![M]\!]_{\Downarrow} = \sup\{\mathscr{D} \mid M \Downarrow_n \mathscr{D}\}$$

It would be quite disappointing if the two object above were different. Indeed, this section is devoted to proving the following theorem:

THEOREM 2. *For every term M, $[\![M]\!]_{\Rightarrow} = [\![M]\!]_{\Downarrow}$.*

Theorem 2 can be proved by showing that any big-step approximation can itself over-approximated with small-step, and vice versa. Let us start by showing that, essentially, the big-step rule for applications is small-step-admissible:

LEMMA 15. *If $M \Rightarrow_n \mathscr{D}$, $N \Rightarrow_m \mathscr{E}$, and for all L and V, $L\{V/x\} \Rightarrow_p \mathscr{E}_{L,V}$, then $MN \Rightarrow_{n+m+p} \mathscr{F}$ such that for all A*

$$\begin{aligned}\mathscr{F}(A) \geq\ & \mathscr{D}|_{\{\mathtt{fail}\}}(A) + \mathscr{D}(\mathbb{R}) \cdot [\mathtt{fail} \in A] \\ & + \mathscr{D}(\mathcal{V}_\lambda) \cdot \mathscr{E}|_{\{\mathtt{fail}\}}(A) \\ & + \iint \mathscr{E}_{L,V}(A)\, \mathscr{D}|_{\mathcal{V}_\lambda}(\lambda x.dL)\, \mathscr{E}|_{\mathcal{V}}(dV).\end{aligned}$$

LEMMA 16. *If $M \Downarrow_n \mathscr{D}$ there is $\mathscr{E}$ s.t. $M \Rightarrow_{3^n} \mathscr{E}$ and $\mathscr{E} \geq \mathscr{D}$.*

PROOF. By induction on n. The only interesting case is when M is an application, and there we simply use Lemma 15. □

At this point, we already know that $[\![M]\!]_{\Rightarrow} \geq [\![M]\!]_{\Downarrow}$. The symmetric inequality can be proved by showing that the big-step rule for applications can be *inverted* in the small-step:

$$\frac{n>0}{G \Downarrow_n \delta(G)}\ (\text{DEVAL VAL}) \qquad \frac{}{M \Downarrow_0 \mathbf{0}}\ (\text{DEVAL EMPTY}) \qquad \frac{n>0}{T \Downarrow_n \delta(\texttt{fail})}\ (\text{DEVAL FAIL})$$

$$\frac{n>0}{\mathsf{D}(\vec{c}) \Downarrow_n \mu_{\mathsf{D}(\vec{c})}}\ (\text{DEVAL SAMP}) \qquad \frac{n>0}{g(\vec{c}) \Downarrow_n \delta(\sigma_g(\vec{c}))}\ (\text{DEVAL FUN}) \qquad \frac{0<c\leq 1 \qquad n>0}{\texttt{score}(c) \Downarrow_n c \cdot \delta(\texttt{true})}\ (\text{DEVAL SCORE})$$

$$\frac{M \Downarrow_n \mathscr{D}}{\texttt{if true then } M \texttt{ else } N \Downarrow_{n+1} \mathscr{D}}\ (\text{DEVAL IF TRUE}) \qquad \frac{N \Downarrow_n \mathscr{D}}{\texttt{if false then } M \texttt{ else } N \Downarrow_{n+1} \mathscr{D}}\ (\text{DEVAL IF FALSE})$$

$$\frac{M \Downarrow_n \mathscr{D} \qquad N \Downarrow_n \mathscr{E} \qquad \{L\{V/x\} \Downarrow_n \mathscr{E}_{L,V}\}_{(\lambda x.L)\in\mathrm{supp}(\mathscr{D}),V\in\mathrm{supp}(\mathscr{E})}}{MN \Downarrow_{n+1} A \mapsto \mathscr{D}|_{\{\texttt{fail}\}}(A) + \mathscr{D}(\mathbb{R})[\texttt{fail} \in A] + \mathscr{D}(\mathcal{V}_\lambda)\cdot\mathscr{E}|_{\{\texttt{fail}\}}(A) + \iint \mathscr{E}_{L,V}(A)\,\mathscr{D}|_{\mathcal{V}_\lambda}(\lambda x.dL)\,\mathscr{E}|_{\mathcal{V}}(dV)}\ (\text{DEVAL APPL})$$

Figure 7. Step Indexed Approximation Big-Step Semantics.

LEMMA 17. *If $MN \Rightarrow_{n+1} \mathscr{D}$, then $M \Rightarrow_n \mathscr{E}$, $N \Rightarrow_n \mathscr{F}$ and for all P and V, $P\{V/x\} \Rightarrow_n \mathscr{G}_{P,V}$ such that for all A,*

$$\mathscr{D}(A) \leq \mathscr{E}|_{\{\texttt{fail}\}}(A) + \mathscr{E}(\mathbb{R})\cdot[\texttt{fail}\in A] + \mathscr{E}(\mathcal{V}_\lambda)\cdot\mathscr{F}|_{\{\texttt{fail}\}}(A) + \iint \mathscr{G}_{P,V}(A)\,\mathscr{E}|_{\mathcal{V}_\lambda}(\lambda x.dP)\,\mathscr{F}|_{\mathcal{V}}(dV).$$

LEMMA 18. *If $M \Rightarrow_n \mathscr{D}$, then there is $\mathscr{E}$ such that $M \Downarrow_n \mathscr{E}$ and $\mathscr{E} \geq \mathscr{D}$.*

PROOF. Again, this is an induction on n that makes essential use, this time, of Lemma 17. □

RESTATEMENT OF THEOREM 2. *For all M, $[\![M]\!]_\Rightarrow = [\![M]\!]_\Downarrow$.*

PROOF. This is a consequence of Lemma 16 and Lemma 18. □

In subsequent sections we let $[\![M]\!]$ stand for $[\![M]\!]_\Rightarrow$ or $[\![M]\!]_\Downarrow$.

4.5 Geometric Distribution, Revisited

Let's consider again the geometric distribution of Section 2.4. There is a monotonically increasing map $f : \mathbb{N} \to \mathbb{N}$ such that for every n, it holds that

$$(geometric\ 0.5) \Rightarrow_{f(n)} \sum_{i=0}^{n} \frac{1}{2^{i+1}}\delta(i)$$

As a consequence, $[\![geometric\ 0.5]\!] = \sum_{i=0}^{\infty} \frac{1}{2^{i+1}}\delta(i)$.

4.6 Distribution-based and Sampling-based Semantics are Equivalent

This section is a proof of the following theorem.

THEOREM 3. *For every term M, $[\![M]\!]_{\mathbb{S}} = [\![M]\!]$.*

The way to prove Theorem 3 is by looking at traces of *bounded* length. For every $n \in \mathbb{N}$, let $\mathbb{S}_n$ be the set of sample traces of length at most n. We define the result distribution $[\![M]\!]^n_{\mathbb{S}}$ as follows:

$$[\![M]\!]^n_{\mathbb{S}}(A) = \int_{\mathbb{S}_n} \mathbf{P}_M(s)\cdot[\mathbf{O}_M(s)\in A]\,ds$$

The integral over *all* traces can be seen as the limit of all integrals over bounded-length traces:

LEMMA 19. *If $f : \mathbb{S} \to \mathbb{R}_+$ is measurable then $\int f = \sup_n \int_{\mathbb{S}_n} f$.*

PROOF. Let $g_n(s) = f(s)\cdot[s \in \mathbb{S}_n]$, so $\int_{\mathbb{S}_n} f = \int g_n$ by definition. Since the g_n are converging to f pointwise from below, we have $\int f = \sup_n \int g_n$ by the monotone convergence theorem. □

A corollary is that $[\![M]\!]_{\mathbb{S}} = \sup_{n\in\mathbb{N}} [\![M]\!]^n_{\mathbb{S}}$. The following is a useful technical lemma.

LEMMA 20. $[\![E[\mathsf{D}(\vec{c})]]\!]^{n+1}_{\mathbb{S}}(A) = \int [\![N]\!]^n_{\mathbb{S}}(A)\ E\{\mu_{\mathsf{D}}(\vec{c})\}(dN)$.

A program M is said to *deterministically diverge* iff $(M, 1, s) \Rightarrow (N, w, [])$ implies that $w = 1$, $s = []$, and N is *not* a generalized value. Terms that deterministically diverge have very predictable semantics, both distribution- and sampling-based.

LEMMA 21. *If M deterministically diverges then $[\![M]\!] = [\![M]\!]_{\mathbb{S}} = \mathbf{0}$.*

A program M is said to *deterministically converge to a program N* iff $(M, 1, []) \Rightarrow (N, 1, [])$. Any term that deterministically converges to another term has the same semantics as the latter.

LEMMA 22. *Let M deterministically converge to N. Then:*
- *$\mathscr{D} \leq \mathscr{E}$ whenever $M \Rightarrow_n \mathscr{D}$; and $N \Rightarrow_n \mathscr{E}$;*
- *$[\![M]\!]^n_{\mathbb{S}} = [\![N]\!]^n_{\mathbb{S}}$;*
- *$[\![M]\!] = [\![N]\!]$ and $[\![M]\!]_{\mathbb{S}} = [\![N]\!]_{\mathbb{S}}$*

If a term does not diverge deterministically, then it converges either to a generalized value or to a term that performs a sampling.

LEMMA 23. *For every program M, exactly one of the following conditions holds:*
- *M deterministically diverges;*
- *There is generalized value G such that M deterministically converges to G*
- *There are $E, \mathsf{D}, c_1, \ldots, c_{|\mathsf{D}|}$ such that M deterministically converges to $E[\mathsf{D}(c_1, \ldots, c_{|\mathsf{D}|})]$.*

We are finally ready to give the two main lemmas that lead to a proof of Theorem 3. The first one tells us that any distribution-based approximation is smaller than the sampling based semantics:

LEMMA 24. *If $M \Rightarrow_n \mathscr{D}$, then $\mathscr{D} \leq [\![M]\!]_{\mathbb{S}}$.*

PROOF. By induction on n:
- If $n = 0$, then $\mathscr{D}$ is necessarily $\mathbf{0}$, and we are done.
- About the inductive case, let's distinguish three cases depending on the three cases of Lemma 23, applied to M:
  - If M deterministically diverges, then by Lemma 21, $\mathscr{D} \leq [\![M]\!] = [\![M]\!]_{\mathbb{S}}$.
  - If M deterministically converges to a generalized value G, then by Lemma 22, it holds that
  $$\mathscr{D} \leq [\![M]\!] = [\![G]\!] = \delta(G) = [\![G]\!]_{\mathbb{S}} = [\![M]\!]_{\mathbb{S}}.$$
  - If M deterministically converges to $E[\mathsf{D}(\vec{c})]$, let $\mathscr{E}$ be such that $E[\mathsf{D}(\vec{c})] \Rightarrow_{n+1} \mathscr{E}$. By Lemma 22 and Lemma 20 we have, by induction hypothesis, that
  $$\begin{aligned}\mathscr{D}(A) \leq \mathscr{E}(A) &= \int \mathscr{F}_N(A)\ E\{\mu_{\mathsf{D}}(\vec{c})\}(dN)\\ &\leq \int [\![N]\!]_{\mathbb{S}}(A)\ E\{\mu_{\mathsf{D}}(\vec{c})\}(dN)\\ &= [\![M]\!]_{\mathbb{S}}\end{aligned}$$

where $N \Rightarrow_n \mathscr{F}_N$. □

The second main lemma tells us that if we limit our attention to traces of length at most n, then we stay below distribution-based semantics:

LEMMA 25. *For every $n \in \mathbb{N}$, $[\![M]\!]^n_{\mathbb{S}} \leq [\![M]\!]$.*

PROOF. By induction on n, following the same schema as the proof of Lemma 24. It can be found in Borgström et al. (2015). □

RESTATEMENT OF THEOREM 3. $[\![M]\!]_{\mathbb{S}} = [\![M]\!]$.

PROOF.

$$\begin{aligned}
[\![M]\!]_{\Rightarrow} &= \sup_{n\in\mathbb{N}}\{\mathscr{D} \mid M \to_n \mathscr{D}\} && \text{(by definition)}\\
&\leq [\![M]\!]_{\mathbb{S}} && \text{(by Lemma 24)}\\
&= \sup_{n\in\mathbb{N}} [\![M]\!]^n_{\mathbb{S}} && \text{(by Lemma 19)}\\
&\leq [\![M]\!] && \text{(by Lemma 25)}\\
&= [\![M]\!]_{\Rightarrow} && \text{(by Theorem 2)}
\end{aligned}$$ □

COROLLARY 1. *The measures $\langle\!\langle M \rangle\!\rangle$ and $\langle\!\langle M \rangle\!\rangle^{\mathcal{V}}$ are sub-probability distributions.*

4.7 An Application of the Distribution-Based Semantics

We can easily prove the equation $[\![\texttt{score}(V)]\!]|_{\mathcal{V}} = [\![MV]\!]|_{\mathcal{V}}$ for all values V, where M is the term

$$\begin{aligned}
&\lambda x.\texttt{if } (0 < x) \wedge (x \leq 1)\\
&\quad \texttt{then } (\texttt{if } \mathit{flip}(x) \texttt{ then true else fail})\\
&\quad \texttt{else fail}
\end{aligned}$$

This shows that even though $\texttt{score}(V)$ and MV *do not* have the same sampling-based semantics, they can be used interchangeably whenever only their extensional, distribution-based behaviour on values is important. We use the equation to our advantage by encoding soft constraints with `score` instead of `flip` (as discussed in Section 2.5), as the fewer the nuisance parameters the better for inference.

4.8 Motivation for 1-Bounded Scores

Recall that we only consider $\texttt{score}(c)$ for $c \in (0, 1]$. Admitting $\texttt{score}(2)$ (say), we exhibit an anomaly by constructing a recursive program that intuitively terminates with probability 1, but where the expected value of its score is infinite. Let

$$\begin{aligned}
\mathit{inflate} := \mathit{fix}\ f\ \lambda x.&\\
&\texttt{if } \mathit{flip}(0.5) \texttt{ then score}(2); (f\ x) \texttt{ else } x.
\end{aligned}$$

Since $[\![\texttt{score}(2)]\!] = [\![\texttt{fail}]\!]$ we have $[\![\mathit{inflate}\ V]\!] = 0.5 \cdot \delta(V) + 0.5 \cdot \delta(\texttt{fail})$ for any V. However, evaluating $\mathit{inflate}\ V$ in a version of our trace semantics where the argument to `score` may be 2 yields

$$\langle\!\langle \mathit{inflate}\ V \rangle\!\rangle(\mathbb{S}_n) = \sum_{k=1}^{n} 1/2 = n/2$$

and so there $[\![\mathit{inflate}\ V]\!]_{\mathbb{S}}(A) = \infty$ if $V \in A$, otherwise 0.

This example shows that even statically bounded scores in combination with recursion may yield return value measures that are not even σ-finite, causing many standard results in measure theory not to apply. For this reason, we restrict attention to positive scores bounded by one. An alternative approach would be to admit unbounded scores, and restrict attention to those programs for which $[\![M]\!]_{\mathbb{S}}(\mathcal{V}) < \infty$.

5. Inference

In this section, we present a variant of the Metropolis-Hastings (MH) algorithm (Metropolis et al. 1953; Hastings 1970) for sampling the return values of a particular closed term $M \in C\Lambda$. This algorithm yields consecutive samples from a Markov chain over $\mathbb{S}$, such that the density of the samples s converges to $\mathbf{P}^{\mathcal{V}}_M(s)$ up to normalization. We can then apply the function $\mathbf{O}_M$ to obtain the return value of M for a given trace.

We prove correctness of this algorithm by showing that as the number of samples goes to infinity, the distribution of the samples approaches the distributional semantics of the program.

5.1 A Metropolis-Hastings Sampling Algorithm

We begin by outlining a generic Metropolis-Hastings algorithm for probabilistic programs, parametric in a proposal density function $q(s, t)$. The algorithm consists of three steps:

1. Pick an initial state s with $\mathbf{P}^{\mathcal{V}}_M(s) \neq 0$ (e.g., by running M).
2. Draw the next state t at random with probability density $q(s, t)$.
3. Compute α as below.

$$\alpha = \min\left(1, \frac{\mathbf{P}^{\mathcal{V}}_M(t)}{\mathbf{P}^{\mathcal{V}}_M(s)} \cdot \frac{q(t, s)}{q(s, t)}\right) \tag{1}$$

- With probability α, output t and repeat from 2 with $s := t$.
- Otherwise, output s and repeat from 2 with s unchanged.

The formula used for the number α above is often called the Hastings *acceptance probability*. Different probabilistic programming language implementations use different choices for the density q above, based on pragmatics. The trivial choice would be to let $q(s, t) = \mathbf{P}^{\mathcal{V}}_M(t)$ for all s, which always yields $\alpha = 1$ and so is equivalent to rejection sampling. We here define another simple density function q (based on Hur et al. (2015)), giving emphasis to the conditions that it needs to satisfy in order to prove the convergence of the Markov chain given by the Metropolis-Hastings algorithm (Theorem 4).

5.2 Proposal Density

In the following, let M be a fixed program. Given a trace $s = [c_1, \ldots, c_n]$, we write $s_{i..j}$ for the trace $[c_i, \ldots, c_j]$ when $1 \leq i \leq j \leq n$. Intuitively, the following procedure describes how to obtain the proposal kernel density (q above):

1. Given a trace s of length n, let $t = [t_1, \ldots, t_n]$ where each t_i is drawn independently from a normal distribution with mean s_i and variance σ^2, and let p_i be the probability density of t_i.
2. Let $k \leq n$ be the largest number such that $(M, 1, t_{1..k}) \Rightarrow (M', w, [])$. There are three cases:
   - If $k = n$, run $M' \Downarrow^{w'}_{t'} V$, and let $q(s, t@t') = p_1 \ldots p_n w'$.
   - If $k < n$ and $M' \Downarrow^{1}_{[]} V$, let $q(s, t_{1..k}) = p_1 \ldots p_k$.
   - Otherwise, let $q(s, t_{1..k}) = 0$ and propose the trace $[]$.

To define this density formally, we first give a function that partially evaluates M given a trace. Let `peval` be a function taking a closed term M and trace s and returning the closed term M' obtained after applying just as many reduction steps to M as required to use up the entire trace s (or `fail` if this cannot be done).

$$\texttt{peval}(M, s) = \begin{cases} M & \text{if } s = [] \\ M' & \text{if } (M, 1, s) \Rightarrow (M_k, w_k, s_k) \to (M', w', []) \\ & \text{for some } M_k, w_k, s_k, w' \text{ such that } s_k \neq [] \\ \texttt{fail} & \text{otherwise} \end{cases}$$

5.3 A Metropolis-Hastings Proposal Kernel

We define the transition kernel $Q(s, A)$ of the Markov chain constructed by the algorithm by integrating a density $q(s, t)$ (as a func-

$$
\begin{aligned}
q(s,t) &= (\Pi_{i=1}^{k} \mathrm{pdf}_{\mathsf{Gaussian}}(s_i, \sigma^2, t_i)) \cdot \mathbf{P}_N^{\mathcal{V}}(t_{k+1..|t|}) \\
&\quad \text{if } |t| \neq 0, \text{ where } k = \min\{|s|, |t|\} \\
&\quad \text{and } N = \mathtt{peval}(M, t_{1..k}) \\
q(s,[]) &= 1 - \int_A q(s,t)\, dt, \text{ where } A = \{t \mid |t| \neq 0\} \\
Q(s,A) &= \int_A q(s,t)\, dt
\end{aligned}
$$

Figure 8. Proposal Density $q(s,t)$ and Kernel $Q(s,A)$ for Program M

tion of t) over A with respect to the stock measure μ on program traces. For technical reasons, we need to ensure that Q is a probability kernel, i.e., that $Q(s,\mathbb{S}) = 1$ for all s. We normalize $q(s,\cdot)$ by giving non-zero probability $q(s,[])$ to transitions ending in $[]$ (which is not a completed trace of M by assumption). All this is in Figure 8.

The integral $\int_A q(s,t)dt$ is well-defined if and only if $q(s,\cdot)$ is non-negative and measurable for every s. In order to show that this property is satisfied, we first need to prove that the `peval` function, used in the definition of q, is measurable:

LEMMA 26. `peval` *is a measurable function* $C\Lambda \times \mathbb{S} \to C\Lambda$.

Using this result, we can show that q, as a function defined on pairs of traces, is measurable.

LEMMA 27. *For any program M, the transition density $q(\cdot,\cdot) : (\mathbb{S} \times \mathbb{S}) \to \mathbb{R}_+$ is measurable.*

By a well-known result in measure theory (Billingsley 1995, Theorem 18.1), it follows that $q(s,\cdot)$ is measurable for every $s \in \mathbb{S}$. To define the transition kernel for the algorithm in terms of the proposal kernel Q, we need to show that Q is a probability kernel.

LEMMA 28. *The function Q is a probability kernel on $(\mathbb{S}, \mathcal{S})$.*

The proofs of lemmas 26, 27 and 28 can be found in the long version of this paper (Borgström et al. 2015).

5.4 Transition Kernel of the Markov Chain

We now use the proposal kernel Q to construct the transition kernel of the Markov chain induced by the algorithm. To avoid trivial cases, we assume that M has positive success probability and does not behave deterministically, i.e., that $[\![M]\!](\mathcal{V}) > 0$ and $\langle\!\langle M \rangle\!\rangle(\{[]\}) = 0$.

Hastings' Acceptance Probability α is defined as in Equation (1) on page 10, where we let $\alpha(s,t) = 0$ if $\mathbf{P}_M^{\mathcal{V}}(t) = 0$ and otherwise $\alpha(s,t) = 1$ if $\mathbf{P}_M^{\mathcal{V}}(s) \cdot q(s,t) = 0$. Given the proposal transition kernel Q and the acceptance ratio α, the Metropolis-Hastings algorithm yields a Markov chain over traces with the following transition probability kernel.

$$
P(s,A) = \int_A \alpha(s,t)\, Q(s,dt) + [s \in A] \cdot \int (1 - \alpha(s,t))\, Q(s,dt). \tag{2}
$$

Define $P^n(s,A)$ to be the probability of the n:th element of the chain with transition kernel P starting at s being in A:

$$
\begin{aligned}
P^0(s,A) &= [s \in A] \\
P^{n+1}(s,A) &= \int P(t,A) P^n(s,dt)
\end{aligned}
$$

LEMMA 29. *If $s_0 \in \mathbf{O}_M^{-1}(\mathcal{V})$ then $P^n(s_0, \mathbf{O}_M^{-1}(\mathcal{V})) = 1$.*

PROOF. By induction on n. □

LEMMA 30. *There is $0 \leq c < 1$ such that $P^n([], \mathbf{O}_M^{-1}(\mathcal{V})) = 1 - c^n$ and $P^n([], \{[]\}) = c^n$.*

PROOF. Let $c = 1 - \langle\!\langle M \rangle\!\rangle^{\mathcal{V}}(\mathbb{S} \setminus \{[]\})$. By assumption $[] \notin \mathbf{O}_M^{-1}(\mathcal{V})$ and $c < 1$, and since $\langle\!\langle M \rangle\!\rangle^{\mathcal{V}}$ is a sub-probability distribution we have $0 \leq c$. We proceed by induction on n. The base case is trivial. For the induction case, we have $P(s, \mathbb{S} \setminus \{[]\}) = 1$ for all $s \in \mathbf{O}_M^{-1}(\mathcal{V})$. Finally

$$
\begin{aligned}
P([], \mathbf{O}_M^{-1}(\mathcal{V})) &= \int_{\mathrm{supp}(\mathbf{P}_M^{\mathcal{V}})} \mathbf{P}_M^{\mathcal{V}} = \\
&\quad \langle\!\langle M \rangle\!\rangle^{\mathcal{V}}(\mathbf{O}_M^{-1}(\mathcal{V})) = \langle\!\langle M \rangle\!\rangle^{\mathcal{V}}(\mathbb{S} \setminus \{[]\}). \; \square
\end{aligned}
$$

Based on Lemma 29 and 30, we below consider the Markov chain with kernel P restricted to $\mathbf{O}_M^{-1}(\mathcal{V}) \cup \{[]\}$.

5.5 Correctness of Inference

By saying that the inference algorithm is correct, we mean that as the number of steps goes to infinity, the distribution of generated samples approaches the distribution specified by the sampling-based semantics of the program.

Formally, we define $T^n(s,A) = P^n(s, \mathbf{O}_M^{-1}(A))$ as the value sample distribution at step n of the Metropolis-Hastings Markov chain. For two measures defined on the same measurable space (X, Σ), we also define the variation norm $||\mu_1 - \mu_2||$ as:

$$
||\mu_1 - \mu_2|| = \sup_{A \in \Sigma} |\mu_1(A) - \mu_2(A)|
$$

We want to prove the following theorem:

THEOREM 4 (Correctness). *For every trace s with $\mathbf{P}_M^{\mathcal{V}}(s) \neq 0$,*

$$
\lim_{n \to \infty} ||T^n(s,\cdot) - [\![M]\!]|_{\mathcal{V}}|| = 0.
$$

To do so, we first need to investigate the convergence of P^n to our target distribution π, defined as follows:

$$
\pi(A) = \langle\!\langle M \rangle\!\rangle^{\mathcal{V}}(A) / \langle\!\langle M \rangle\!\rangle^{\mathcal{V}}(\mathbb{S}).
$$

We use a sequence of known results for Metropolis-Hastings Markov chains (Tierney 1994) to prove that P^n converges to π. We say that a Markov chain transition kernel P is *$\mathscr{D}$-irreducible* if $\mathscr{D}$ is a non-zero sub-probability distribution on $(\mathbb{S}, \mathcal{S})$, and for all $x \in \mathbb{S}, A \in \mathcal{S}$ there exists an integer $n > 0$ such that $\mathscr{D}(A) > 0$ implies $P^n(x,A) > 0$. We say that P is *$\mathscr{D}$-aperiodic* if there do not exist $d \geq 2$ and disjoint $B_1, \ldots, B_d$ such that $\mathscr{D}(B_1) > 0$, and $x \in B_d$ implies $P(x, B_1) = 1$, and $x \in B_i$ implies that $P(x, B_{i+1}) = 1$ for $i \in \{1, \ldots, d-1\}$.

LEMMA 31 (Tierney (1994), Theorem 1 and Corollary 2). *Let K be the transition kernel of a Markov chain given by the Metropolis-Hastings algorithm with target distribution $\mathscr{D}$. If K is $\mathscr{D}$-irreducible and aperiodic, then for all s, $\lim_{n \to \infty} ||K^n(s,\cdot) - \mathscr{D}|| = 0$.*

LEMMA 32 (Strong Irreducibility). *If $\mathbf{P}_M^{\mathcal{V}}(s) > 0$ and $\langle\!\langle M \rangle\!\rangle^{\mathcal{V}}(A) > 0$ then $P(s,A) > 0$.*

PROOF. There is n such that $\mathbf{P}_M^{\mathcal{V}}(A \cap \mathbb{S}_n) > 0$. Write $A|_n = A \cap \mathbb{S}_n$. For all $t \in A|_n$, $q(s,t) > 0$ by case analysis on whether

$n \leq |s|$. Since $\mu(A|_n) > 0$ and $\mathbf{P}_M^{\mathcal{V}}(t) > 0$ for all $t \in A|_n$,

$$\begin{aligned} P(s,A) &\geq P(s,A|_n) \\ &\geq \int_{A|_n} \alpha(s,t)\, Q(s,dt) \\ &= \int_{A|_n} \alpha(s,t) q(s,t)\, dt \\ &= \int_{A|_n} \min\{q(s,t), \frac{\mathbf{P}_M^{\mathcal{V}}(t) q(t,s)}{\mathbf{P}_M^{\mathcal{V}}(s)}\}\, dt \\ &> 0. \end{aligned}$$

□

COROLLARY 2 (Irreducibility). *P as given by Equation (2) is π-irreducible.*

LEMMA 33 (Aperiodicity). *P as given by Equation (2) is π-aperiodic.*

PROOF. Assume that B_1, B_2 are disjoint sets such that $\pi(B_1) > 0$ and $P(s, B_2) = 1$ for all $s \in B_1$. If $s \in B_1$, Lemma 32 gives that $P(s, B_1) > 0$, so $P(s, B_2) < P(s, \mathbb{S}) = 1$, which is a contradiction. A fortiori, P is π-aperiodic. □

LEMMA 34. *If μ_1 and μ_2 are measures on (X_1, Σ_1) and $f : X_1 \to X_2$ is measurable Σ_1/Σ_2, then*

$$||\mu_1 f^{-1} - \mu_2 f^{-1}|| \leq ||\mu_1 - \mu_2||$$

RESTATEMENT OF THEOREM 4. *For every trace s with $\mathbf{P}_M^{\mathcal{V}}(s) \neq 0$,*

$$\lim_{n\to\infty} ||T^n(s,\cdot) - [\![M]\!]|_{\mathcal{V}}|| = 0.$$

PROOF. By Corollary 2, P is π-irreducible, and by Lemma 33, P is π-aperiodic. Lemma 31 then yields that

$$\lim_{n\to\infty} ||P^n(x,\cdot) - \pi|| = 0.$$

By definition, $T^n(s,A) = P^n(s, \mathbf{O}_M^{-1}(A))$ and $[\![M]\!]|_{\mathcal{V}}(A) = [\![M]\!](A \cap \mathcal{V})/[\![M]\!](\mathcal{V})$. By Theorem 3, $[\![M]\!](A \cap \mathcal{V}) = [\![M]\!]_{\mathbb{S}}(A \cap \mathcal{V}) = \langle\!\langle M \rangle\!\rangle(\mathbf{O}_M^{-1}(A \cap \mathcal{V}) = \langle\!\langle M \rangle\!\rangle(\mathbf{O}_M^{-1}(A) \cap \mathbf{O}_M^{-1}(\mathcal{V})) = \langle\!\langle M \rangle\!\rangle^{\mathcal{V}}(\mathbf{O}_M^{-1}(A))$ and similarly $[\![M]\!](\mathcal{V}) = \langle\!\langle M \rangle\!\rangle(\mathbf{O}_M^{-1}(\mathcal{V})) = \langle\!\langle M \rangle\!\rangle^{\mathcal{V}}(\mathbb{S})$, which gives $[\![M]\!]|_{\mathcal{V}}(A) = \pi((O_M^{-1}(A))$. Thus, by Lemma 34 and the squeeze theorem for limits we get

$$\lim_{n\to\infty} ||T^n(s,\cdot) - [\![M]\!]|_{\mathcal{V}}|| \leq \lim_{n\to\infty} ||P^n(s,\cdot) - \pi|| = 0.$$ □

5.6 Examples

To illustrate how inference works, we revisit the geometric distribution and linear regression examples from Section 2. Before discussing the transition kernels for these models, note that the products of Gaussian densities always cancel out in the acceptance probability α, because $\mathrm{pdf}_{\mathsf{Gaussian}}(s_i, \sigma^2, t_i) = \mathrm{pdf}_{\mathsf{Gaussian}}(t_i, \sigma^2, s_i)$ by the definition of the Gaussian PDF.

Geometric Distribution Let us begin with the implementation of the geometric distribution, described in 2.4, which we will call M_1. Since the only random primitive used in M_1 is rnd, whose density is 1 on all its support, and there are no calls to `score`, the weight of any trace that yields a value must be 1. Because the *geometric* function applied to 0.5 returns a value immediately when the call to rnd returns a number smaller than 0.5, and recursively calls itself otherwise, otherwise returns a value immediately

The function *geometric* applied to 0.5 calls itself recursively if the call to rnd returned a value greater or equal to a half, and returns a value immediately otherwise. Hence, every valid trace consists of a sequence of numbers in $[0.5, 1]$, followed by a number in $[0, 0.5)$, and so the set of valid traces is $S_1 = \{s \mid s_i \in [0, 0.5)$ for $i < |s| \wedge s_{|s|} \in [0.5, 1] \wedge |s| > 0\}$. The proposal density is

$$q(s,t) = [t \in S_1] \Pi_{i=1}^{k} \mathrm{pdf}_{\mathsf{Gaussian}}(s_i, \sigma^2, t_i)$$

where $k = \min\{|s|, |t|\}$. The term $[t \in S_1]$ reflects the fact that for every non-valid trace, $\mathbf{P}^{\mathcal{V}}_{\mathtt{peval}(M, t_{1..k})}(t_{k+1..|t|}) = 0$.

As noted above, the Gaussians cancel out in the acceptance ratio, and the density of every valid trace is 1, so $\alpha(s,t) = [t \in S_1]$. This means that every valid trace is accepted.

Linear Regression with `flip` Now, consider the linear regression model from section 2.6, which can be translated from Church to the core calculus by applying the rules in Figure 2 (details are omitted, but the translation is straightforward as there is no recursion).

In every trace in this translated model, which we call M_2, we have two draws from $\mathsf{Gaussian}(0, 2)$, whose values are assigned to variables m and b. They are followed by four calls to `rnd` made while evaluating the four calls to `softeq`. The conditioning statement at the end sets the return value to `false` if at least one call to `softeq` evaluates to `false`. Since `softeq` x y returns `true` if and only if the corresponding call to `rnd` returned a value less than `squash` x y, it follows from the definitions of `squash` and `f` that the element of the trace consumed by `softeq` $(f\ x)$ y must be in the interval $[1, \frac{1}{e^{(m\cdot x + b - y)^2}})]$. Note that since the pdf of `rnd` is flat, the weight of any trace depends only on the first two random values, drawn from the Gausians, as long as the remaining four random values are in the right intervals.

The full density for this model is

$$\begin{aligned} \mathbf{P}^{\mathcal{V}}_{M_2}(s) \quad = \quad & \left(\Pi_{i=1}^{2} \mathrm{pdf}_{\mathsf{Gaussian}}(0, 2, s_i)\right) \cdot \\ & \left(\Pi_{i=1}^{4} \left[s_{i+2} \in \left[1, \frac{1}{e^{(s_i \cdot x_i + s_2 - y_i)^2}}\right)\right]\right) \end{aligned}$$

if $s \in \mathbb{R}^6$ and $\mathbf{P}^{\mathcal{V}}_{M_2}(s) = 0$ if $s \notin \mathbb{R}^6$.

The partial derivative of $\mathbf{P}^{\mathcal{V}}_{M_2}(s)$ with respect to each of s_3, s_4, s_5, s_6 is zero wherever defined, precluding the use of efficient gradient-based methods for searching over these components of the trace.

Now, let us derive the density $q(s,t)$, assuming that $\mathbf{P}^{\mathcal{V}}_{M_2}(s) > 0$ (which implies $s \in \mathbb{R}^6$, as shown above) and $t \in \mathbb{R}^6$. Since we have $|s| = |t| = 6$, the formula for q reduces to:

$$q(s,t) = (\Pi_{i=1}^{6} \mathrm{pdf}_{\mathsf{Gaussian}}(s_i, \sigma^2, t_i)) \mathbf{P}^{\mathcal{V}}_{M_2'}([])$$

where $M_2' = \mathtt{peval}(M_2, t)$. Because there cannot be more than six random draws in any run of the program, M_2' is determinstic. This means that $\mathbf{P}^{\mathcal{V}}_{M_2'}([]) = 0$ if $M_2' \Downarrow_1^{[]} \mathtt{fail}$ and $\mathbf{P}^{\mathcal{V}}_{M_2'}([]) = 1$ if $M_2' \Downarrow_1^{[]} \mathtt{fail}$ for some V.

It is easy to check that if $t \notin \mathbb{R}^6$, then $q(s,t) = 0$—since there is no trace of length other than 6 leading to a value, the value of $\mathbf{P}^{\mathcal{V}}_{M_2'}(t_{k+1..|t|})$ in the definition of q must be 0 in this case.

Thus, the proposal density is

$$\begin{aligned} q(s,t) \quad = \quad & \left(\Pi_{i=1}^{6} \mathrm{pdf}_{\mathsf{Gaussian}}(s_i, \sigma^2, t_i)\right) \cdot \\ & \left(\Pi_{i=1}^{4} [t_{i+2} \in [0, \frac{1}{e^{(t_i \cdot x_1 + t_2 - y_i)^2}})]\right) \end{aligned}$$

for $t \in \mathbb{R}^6$ and $q(s,t) = 0$ for $t \notin \mathbb{R}^6$.

Hence, the acceptance ratio reduces to

$$\begin{aligned} \alpha(s,t) \quad = \quad & \min\{1, \frac{\Pi_{i=1}^{2} \mathrm{pdf}_{\mathsf{Gaussian}}(0, 2, t_i)}{\Pi_{i=1}^{2} \mathrm{pdf}_{\mathsf{Gaussian}}(0, 2, s_i)} \cdot \\ & \Pi_{i=1}^{4} [t_{i+2} \in [0, \frac{1}{e^{(t_1 \cdot x_i + t_2 - y_i)^2}})]\} \end{aligned}$$

if $t \in \mathbb{R}^6$ and $\alpha(s,t) = 0$ otherwise.

Note that $\alpha(s,t)$ is only positive if each of t_3, t_4, t_5, t_6 are within a certain (small) interval. This is problematic for an implementation, since it will need to find suitable values for all these components of the trace for every new trace to be proposed, leading to inefficiencies due to a slowly mixing Markov chain.

Linear Regression with score In this alternative version M_3 of the previous model, we also have two draws from $\mathsf{Gaussian}(0,2)$ at the beginning, but the calls to flip are replaced with calls to `score`, which multiply the trace density by a positive number without consuming any elements of the trace. Because the support of the Gaussian PDF is $\mathbb{R}$ and there are precisely two random draws (both from Gaussians) in every trace leading to a value, the set of valid traces is $\mathbb{R}^2$. We have $\mathbf{P}^{\mathcal{V}}_{M_3}(s) = \Pi_{i=1}^{2} \operatorname{pdf}_{\mathsf{Gaussian}}(0,2,s_i)$ if $s \in \mathbb{R}^2$ and $\mathbf{P}^{\mathcal{V}}_{M_3}(s) = 0$ otherwise. Assuming $\mathbf{P}^{\mathcal{V}}_{M_2}(s) > 0$ and $t \in \mathbb{R}^2$, we get the proposal density

$$q(s,t) = \Pi_{i=1}^{6} \operatorname{pdf}_{\mathsf{Gaussian}}(s_i, \sigma^2, t_i) \Pi_{i=1}^{4} \frac{1}{e^{(t_i \cdot x_1 + t_2 - y_i)^2}}$$

where $x = [0,1,2,3]$ and $y = [0,1,4,6]$. If $t \notin \mathbb{R}^2$, then $q(s,t) = 0$, because otherwise there would be a trace of length different than 2 leading to a value.

Thus, the acceptance ratio is

$$\alpha(s,t) = \frac{\Pi_{i=1}^{2} \operatorname{pdf}_{\mathsf{Gaussian}}(0,2,t_i)}{\Pi_{i=1}^{2} \operatorname{pdf}_{\mathsf{Gaussian}}(0,2,s_i)} \cdot \frac{\Pi_{i=1}^{4} e^{(t_i \cdot x_1 + t_2 - y_i)^2}}{\Pi_{i=1}^{4} e^{(s_i \cdot x_1 + s_2 - y_i)^2}}$$

if $t \in \mathbb{R}^2$ and $\alpha(s,t) = 0$ otherwise.

In contrast to the previous example, here the acceptance ratio is positive for all proposals, non-zero gradients exist almost everywhere, and there are four fewer nuisance parameters to deal with (one per data point!). This makes inference for this version of the model much more tractable in practice.

6. Related Work

To the best of our knowledge, the only previous theoretical justification for trace MCMC is the recent work by Hur et al. (2015), who show correctness of trace MCMC for the imperative probabilistic language R2 (Nori et al. 2014). Their result does not apply to higher-order languages such as CHURCH or our λ-calculus. Their algorithm is different from ours in that it exploits the explicit storage locations found in imperative programs, keeping one sample trace per location. The authors do not consider measurability. Their proof of correctness only shows that the acceptance ratio α computed by their algorithm matches Hasting's formula: the authors do not prove convergence of the resulting Markov chai, which may depend on the choices of parameters in the algorithm.

Other probabilistic language implementations also use trace MCMC inference, including CHURCH (Goodman et al. 2008), VENTURE (Mansinghka et al. 2014), WEBPPL (Goodman and Stuhlmüller 2014), and ANGLICAN (Tolpin et al. 2015). These works focusing on efficiency and convergence properties, and do not state formal correctness claims for their implementations.

Wingate et al. (2011) give a general program transformation for a probabilistic language to support trace MCMC, with a focus on labelling sample points in order to maximise sample reuse. Extending our trace semantics with such labelling is important future work, given that Kiselyov (2016) points out some difficulties with the transformation and proposes alternatives.

Many recent probabilistic languages admit arbitrary non-negative scores. This is done either by having an explicit `score`-like function, as in WEBPPL (called `factor`), or by observing that a particular value V was drawn from a given distribution $\mathsf{D}(\vec{c})$ (without adding it to the trace), as in WEB CHURCH (written $(\mathsf{D}\ \vec{c}\ V)$) or ANGLICAN (`observe` $(\mathsf{D}\ \vec{c})\ V$). In recent work for a non-recursive λ-calculus with `score`, Staton et al. (2016) note that unbounded scores introduce the possibility of "infinite model evidence errors". As seen in Section 4.8, even statically bounded scores exhibit this problem in the presence of recursion.

Kozen (1979) gives a semantics of imperative probabilistic programs as partial measurable functions from infinite random traces to final states, which serves as the model for our trace semantics. Kozen also proves this semantics equivalent to a domain-theoretic one. Park et al. (2008) give an operational version of Kozen's trace-based semantics for a λ-calculus with recursion, but "do not investigate measure-theoretic properties". Cousot and Monerau (2012) generalise Kozen's trace-based semantics to consider probabilistic programs as measurable functions from a probability space into a semantics domain, and study abstract interpretation in this setting. Toronto et al. (2015) use a pre-image version of Kozen's semantics to obtain an efficient implementation using rejection sampling. Scibior et al. (2015) define a monadic embedding of probabilistic programming in Haskell along the lines of Kozen's semantics; their paper describes various inference algorithms but has no formal correctness results.

Ramsey and Pfeffer (2002) provide a monadic denotational semantics for a first-order functional language with discrete probabilistic choice, and a Haskell implementation of the expectation monad using variable elimination. Bhat et al. (2013) define a denotational semantics based on density functions for a restricted first-order language with continuous distributions. They also present a type system ensuring that a given program has a density.

Jones (1990, Chapter 8) defines operational and domain-theoretic semantics for a λ-calculus with discrete probabilistic choice and a fixpoint construct. Our distribution-based operational semantics generalises Jones's to deal with continuous distributions. Like Kozen's and Jones's semantics, our operational semantics makes use of the partially additive structure on the category of sub-probability kernels (Panangaden 1999) in order to treat programs that make an unbounded number of random draws. Staton et al. (2016) give a domain-theoretic semantics for a λ-calculus with continuous distributions and unbounded `score`, but without recursion. While giving a fully abstract domain theory for probabilistic λ-calculi with recursion is known to be hard (Jones and Plotkin 1989), there have been recent advances using probabilistic coherence spaces (Danos and Ehrhard 2011; Ehrhard et al. 2014) and game semantics (Danos and Harmer 2002), which in some cases are fully abstract. We see no strong obstacles in applying any of these to a typed version of our calculus, but it is beyond the scope of this work. Another topic for future work are methodologies for equivalence checking in the style of logical relations or bisimilarity, which have been recently shown to work well in *discrete* probabilistic calculi (Bizjak and Birkedal 2015).

7. Conclusions and Remarks

As a foundation for probabilistic inference in languages such as CHURCH, we defined a probabilistic λ-calculus with draws from continuous probability distributions and both hard and soft constraints, defined its semantics as distributions on terms, and proved correctness of a trace MCMC inference algorithm via a sampling semantics for the calculus.

We have taken the sample space to be the real numbers, but any complete separable metric space will do. For example, in order to add discrete distributions to the language we can change $\mathbb{S}$ to $\biguplus_{n \in \mathbb{N}} (\mathbb{R} \uplus \mathbb{N})^n$.

Although our emphasis has been on developing theoretical underpinnings, we also implemented our algorithm in F# to help develop our intuitions and indeed to help debug definitions. The algorithm is correct and effective, but not optimized. In future, we aim to extend our proofs to cover more efficient algorithms, inspired by Wingate et al. (2011) and Kiselyov (2016), for example.

References

S. Bhat, J. Borgström, A. D. Gordon, and C. V. Russo. Deriving probability density functions from probabilistic functional programs. In N. Piterman and S. A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7795 of *LNCS*, pages 508–522, 2013.

P. Billingsley. *Probability and Measure*. Wiley-Interscience, third edition, 1995.

A. Bizjak and L. Birkedal. Step-indexed logical relations for probability. In A. M. Pitts, editor, *Proceedings of FoSSaCS 2015*, volume 9034 of *LNCS*, pages 279–294. Springer, 2015.

J. Borgström, U. Dal Lago, A. D. Gordon, and M. Szymczak. A lambda-calculus foundation for universal probabilistic programming (long version). *CoRR*, abs/1512.08990, Dec 2015.

P. Cousot and M. Monerau. Probabilistic abstract interpretation. In *Proceedings of ESOP 2012*, volume 7211 of *LNCS*, pages 166–190. Springer, 2012.

V. Danos and T. Ehrhard. Probabilistic coherence spaces as a model of higher-order probabilistic computation. *Information and Computation*, 209(6):966–991, 2011.

V. Danos and R. Harmer. Probabilistic game semantics. *ACM Transactions on Computational Logic*, 3(3):359–382, 2002.

T. Ehrhard, C. Tasson, and M. Pagani. Probabilistic coherence spaces are fully abstract for probabilistic PCF. In *Proceedings of POPL 2014*, pages 309–320, 2014.

S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

N. D. Goodman. The principles and practice of probabilistic programming. In *Principles of Programming Languages (POPL'13)*, pages 399–402, 2013.

N. D. Goodman and A. Stuhlmüller. The design and implementation of probabilistic programming languages. `http://dippl.org`, 2014.

N. D. Goodman and J. B. Tenenbaum. Probabilistic models of cognition. `http://probmods.org`, 2014.

N. D. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: a language for generative models. In D. A. McAllester and P. Myllymäki, editors, *Proceedings of UAI 2008*, pages 220–229. AUAI Press, 2008.

A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani. Probabilistic programming. In *Future of Software Engineering (FOSE 2014)*, pages 167–181, 2014.

W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970.

M. D. Homan and A. Gelman. The no-u-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, Jan. 2014.

C. Hur, A. V. Nori, S. K. Rajamani, and S. Samuel. A provably correct sampler for probabilistic programs. In P. Harsha and G. Ramalingam, editors, *Proceedings of FSTTCS 2015*, volume 45 of *LIPIcs*, pages 475–488. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

C. Jones. *Probabilistic Non-determinism*. PhD thesis, University of Edinburgh, 1990. Available as Technical Report CST-6390.

C. Jones and G. D. Plotkin. A probabilistic powerdomain of evaluations. In *Proceedings of LICS 1989*, pages 186–195, 1989.

O. Kiselyov. Problems of the lightweight implementation of probabilistic programming, Jan. 2016. Poster at PPS'2016 workshop.

D. Kozen. Semantics of probabilistic programs. In *Proceedings of FOCS 1979*, pages 101–114. IEEE Computer Society, 1979.

C. D. Manning and H. Schütze. *Foundations of statistical natural language processing*. MIT Press, 1999.

V. K. Mansinghka, D. Selsam, and Y. N. Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *CoRR*, abs/1404.0099, 2014.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

A. V. Nori, C. Hur, S. K. Rajamani, and S. Samuel. R2: an efficient MCMC sampler for probabilistic programs. In C. E. Brodley and P. Stone, editors, *Proceedings of AAAI 2014*, pages 2476–2482. AAAI Press, 2014.

P. Panangaden. The category of Markov kernels. *ENTCS*, 22:171–187, 1999. In proceedings of PROBMIV 1998.

P. Panangaden. *Labelled Markov Processes*. Imperial College Press, 2009.

S. Park, F. Pfenning, and S. Thrun. A probabilistic language based upon sampling functions. *ACM Transactions on Programming Languages and Systems*, 31:1, 2008.

J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

N. Ramsey and A. Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *Proceedings of POPL 2002*, pages 154–165, 2002.

S. J. Russell. Unifying logic and probability. *Communications of the ACM*, 58(7):88–97, 2015.

A. Scibior, Z. Ghahramani, and A. D. Gordon. Practical probabilistic programming with monads. In B. Lippmeier, editor, *Proceedings of Haskell 2015*, pages 165–176. ACM, 2015.

S. Staton, H. Yang, C. Heunen, O. Kammar, and F. Wood. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. *CoRR*, abs/1601.04943, 2016.

T. Tao. *An Introduction to Measure Theory*. AMS, 2011.

S. Thrun. *Exploring artificial intelligence in the new millennium*, chapter Robotic mapping: A survey, pages 1–35. Morgan Kaufmann, 2002.

L. Tierney. Markov chains for exploring posterior distributions. *The Annals of Statistics*, 22(4):1701–1728, 1994.

D. Tolpin, J. van de Meent, and F. Wood. Probabilistic programming in Anglican. In A. Bifet, M. May, B. Zadrozny, R. Gavaldà, D. Pedreschi, F. Bonchi, J. S. Cardoso, and M. Spiliopoulou, editors, *Proceedings of ECML PKDD 2015, Part III*, volume 9286 of *LNCS*, pages 308–311. Springer, 2015.

N. Toronto. *Useful Languages for Probabilistic Modeling and Inference*. PhD thesis, Brigham Young University, Provo, UT, 2014. URL `https://www.cs.umd.edu/~ntoronto/papers/toronto-2014diss.pdf`.

N. Toronto, J. McCarthy, and D. V. Horn. Running probabilistic programs backwards. In J. Vitek, editor, *Proceedings of ESOP 2015*, volume 9032 of *LNCS*, pages 53–79. Springer, 2015.

D. Wingate, A. Stuhlmueller, and N. D. Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. *Journal of Machine Learning Research*, 15:770–778, 2011. In proceedings of AISTATS 2011.